

Kofax FraudOne

The Book on CRS

Version: 4.6.0

Date: 2022-10-17

KOFAX

© 2022 Kofax. All rights reserved.

Kofax is a trademark of Kofax, Inc., registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Kofax.

Table of Contents

Preface	6
Related documentation.....	6
Training.....	7
Getting help with Kofax products.....	7
Migration update.....	8
Chapter 1: What is CRS anyway?	9
CRS?.....	9
What does it do?.....	9
How does it work? System architecture.....	9
The FraudOne system.....	10
Queues.....	10
Workflow.....	10
CRS, rules and risk.....	10
Rule management and configuration server.....	11
Wrap up.....	11
Chapter 2: First steps	12
What do you need?.....	12
Start the necessary system components.....	12
Use the editor.....	13
Wrap up.....	14
Chapter 3: A simple automatic and visual signature workflow	15
What do you want to do?.....	15
Create a new rule set.....	16
Set up the queues.....	17
Choose risk indicators.....	18
Default target.....	20
Your first condition.....	21
Make a decision.....	22
Complete the rules.....	24
ASV routing.....	24
VSV routing.....	28
Save it and choose the active rule set.....	30
Wrapping it up.....	32
Wrap up.....	34

Chapter 4: Rule blocks.....	35
When do you need rule blocks?.....	35
Create a rule block.....	35
Add the rule block to another branch.....	48
Modify a rule block.....	56
Unbind a rule block reference.....	59
Remove a rule block reference.....	60
Delete a rule block reference.....	61
Wrap up.....	63
Chapter 5: Score, risk and priorities.....	64
Basics.....	64
Some preparation.....	64
Score.....	70
How much does it hurt?.....	76
Priority.....	78
Wrap up.....	79
Chapter 6: Managing volume.....	80
Risk based decisions.....	80
Global threshold.....	80
Continue?.....	84
Wrap up.....	85
Chapter 7: Random decisions?.....	86
When you don't want to control everything.....	86
Define a new visual verification step.....	87
Use a random risk variable.....	87
Wrap up.....	93
Chapter 8: Planning for change.....	94
Configuration variables.....	94
BNO dependent variables.....	96
Wrap up.....	97
Chapter 9: Add your own queues, engines and indicators.....	98
Add a custom queue.....	98
BNO specific queues.....	99
Define your own risk indicators.....	104
Factoring in external information.....	106
Wrap up.....	107
Chapter 10: Doing it as a parallel process.....	108
What is the goal?.....	108

Set up queues, indicators and other elements.....	110
Split items.....	112
Route items.....	114
Merge items back together.....	116
Use interim variables to merge parallel branches.....	124
Wrapping it up.....	134
Wrap up.....	134
Chapter 11: Best practice.....	136
Gather information.....	136
Planning for errors.....	137
Rule ordering.....	138
Match rates and results.....	139
Names and comments.....	139
Keep it simple.....	139
Related conditions.....	140
Engine only workflows.....	140
Reporting errors.....	141
Wrap up.....	142
Chapter 12: Reference.....	143
Configuration elements.....	143
Queues.....	143
Risk indicators.....	145
Configuration variables.....	150
Reset points.....	151
Interim variables.....	152
Rule elements.....	152
Rule graph.....	152
Conditions.....	153
Decisions.....	154
Rule blocks.....	157
Default target.....	157

Preface

Related documentation

The full documentation set for Kofax FraudOne is available at the following location:

<https://docshield.kofax.com/Portal/Products/FO/4.6.0-e4jy6kf7pr/FO.htm>

In addition to this guide, the documentation set includes the following items:

Release notes

- *Kofax FraudOne Release Notes*

Technical specifications

- *Kofax FraudOne Technical Specifications*

Guides

- *Kofax FraudOne Administrator's Guide*
- *Kofax FraudOne Archive Interface Server*
- *Kofax FraudOne ASV Blackbox*
- *Kofax FraudOne Common API Specifications for GIA Engines*
- *Kofax FraudOne Data Warehouse Installation and Operation Guide*
- *Kofax FraudOne Extended Reporting Features and Statistics*
- *Kofax FraudOne Feature Codes*
- *Kofax FraudOne Global Fraud Signature Web Service Developer's Guide*
- *Kofax FraudOne Installation and Migration Guide*
- *Kofax FraudOne Java Client Customization Guide*
- *Kofax FraudOne Java Client Customization Layer*
- *Kofax FraudOne Report Component Installation Guide*
- *Kofax FraudOne Service Program Configuration*
- *Kofax FraudOne Service Program Interfaces*
- *Kofax FraudOne SignCheck Result Codes*
- *Kofax FraudOne Standard Reporting Features and Statistics*
- *Kofax FraudOne Thin Client Customization Guide*
- *Kofax FraudOne Variant Cleanup Utility*

Help

- *Kofax FraudOne Administration Client Help*
- *Kofax FraudOne Error Messages Help*
- *Kofax FraudOne Java Client Help*
- *Kofax FraudOne Server Monitor Help*
- *Kofax FraudOne Thin Client Help*

Training


Kofax offers both classroom and online training to help you make the most of your product. To learn more about training courses and schedules, visit the [Kofax Education Portal](#) on the Kofax website.

Getting help with Kofax products

The [Kofax Knowledge Base](#) repository contains articles that are updated on a regular basis to keep you informed about Kofax products. We encourage you to use the Knowledge Base to obtain answers to your product questions.

To access the Kofax Knowledge Base:

1. Go to the [Kofax website](#) home page and select **Support**.
2. When the Support page appears, select **Customer Support > Knowledge Base**.

 The Kofax Knowledge Base is optimized for use with Google Chrome, Mozilla Firefox or Microsoft Edge.

The Kofax Knowledge Base provides:

- Powerful search capabilities to help you quickly locate the information you need.
Type your search terms or phrase into the **Search** box, and then click the search icon.
- Product information, configuration details and documentation, including release news.
Scroll through the Kofax Knowledge Base home page to locate a product family. Then click a product family name to view a list of related articles. Please note that some product families require a valid Kofax Portal login to view related articles.

From the Knowledge Base home page, you can:

- Access the Kofax Community (for all customers).
Click the **Community** link at the top of the page.
- Access the Kofax Customer Portal (for eligible customers).
Click the **Support** link at the top of the page. When the Customer & Partner Portals Overview appears, click **Log in to the Customer Portal**.
- Access the Kofax Partner Portal (for eligible partners).

Click the **Support** link at the top of the page. When the Customer & Partner Portals Overview appears, click **Log in to the Partner Portal**.

- Access Kofax support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.

Go to the **General Support** section, click **Support Details**, and then select the appropriate tab.

Migration update

Starting with FraudOne 4.4.1 the amounts used in CRS rules are based on the smallest currency unit (such as cents) instead of full currency (such as dollars).

The change means that the value stored into the amount fields is always an integer value. For example, a value of USD 123,45 (one hundred and twenty three dollars and forty five cents) would be stored in the database as "12345".

The change requires that existing rule graphs which were created with an earlier release than FraudOne 4.4.1 need to be adjusted accordingly. I.e., a CRS rule amount in USD with a value of 100 needs to be adjusted to a value of 10000.

Chapter 1

What is CRS anyway?

CRS?

Ok, so what does CRS really stand for? It means 'Combined Risk Score'.

The engine is designed to combine results of multiple FraudOne and external item processing engines like signature verification, check stock verification, PAD (pre authorized drafts) detection, ASI, visual verification, etc. and score the item based on the evaluation of all these results. In the end an individual item's risk is computed and the item is processed accordingly.

What does it do?

CRS is designed to:

1. Define processing steps (queues) in the FraudOne system.
You can define individual item processing steps and specify the way clients will use them.
2. Evaluate item risk based on the results of the processing steps.
3. Control the way an item is moved from one processing step to another (workflow) based on:
 - Item information (SignCheck)
 - Account information (SignBase)
 - Results from individual processing steps
 - External information
 - Calculated score and risk based on the factors above
 - Provide a final processing recommendation to the banking system

How does it work? System architecture

If you are familiar with the architecture of FraudOne (from experience or reading the *Kofax FraudOne Administrator's Guide*) you can skip to the next chapter.

The FraudOne system

FraudOne is designed to process items (checks). It has two parts: the reference, or account data handling and the item, or check handling. You also call the first SignBase and the later SignCheck.

The purpose of the system is to verify items by comparing the information to the reference data and other available information in several automatic or manual processing steps.

Queues

As opposed to the account reference information, where the FraudOne systems do much to preserve state and history of the information stored, item or check information is transient. It enters the system, is processed by engines and visual verification and leaves the system again with an overall result. At the end of the day DFP (day's final processing) usually cleans away any trace the item might have left.

During its short stay in the FraudOne system, the item passes a number of specific processing steps like ASV (automatic signature verification), APIA (check stock verification), VSV (visual verification), etc. Each of these steps works on a queue. You call it a queue, because items are queued up and wait for the engine or user that works the queue to process them one after another. The order items are picked from the queue is based on item priority.

Workflow

The workflow defines how items move from one queue to another. Items enter the system by being placed into the INPUT queue by the Getter and leave it by being picked up out of the OUTPUT queue by the Putter.

Everything happening in between is controlled by the workflow. In earlier FraudOne versions this used to be a static, customer-specific piece of code that moved the item from one queue to another based on the previous queue result.

With CRS this changed a bit. After the item finishes the processing step in a queue, CRS takes over. It gathers all data necessary to evaluate the item, has a look at it, checks its set of rules and decides where the item needs to go. It then stores the results and moves the item into the next queue according to the decision made, where everything starts again.

This goes on until CRS has enough information to perform the final decision and places the item into the OUTPUT queue.

CRS, rules and risk

The nice thing about this is, that the CRS rules editor gives you control over the rules used to move the item around. The intention of this book is to give you enough information to be able to define your own queues, write your own rules and move the item around the FraudOne system according to your own business needs.

Depending on the information available, you can decide to pay the item, have it reviewed by your staff or reject it. You can score items, compute item risk and present them to analysts with the highest risk items first.

Rule management and configuration server

Rules and CRS configuration are stored by the configuration server inside the system database. You'll need online access to the system to change CRS configuration.

The benefit to this is that rule changes are tracked. You can tell who changed what and when and could also revert to an earlier version. The system can store multiple rule sets and you can activate one or another.

Rules can be moved around by exporting them into an XML file and importing them again. As a matter of fact, if you have questions regarding your rule set, Kofax will be asking you to provide us the exported rule XML so we can review your data with you.

Wrap up

Items are processed by FraudOne one by one.

Items are moved from one processing step to another. Each processing step works on a queue. Items are queued up according to their priority.

They enter the system through the INPUT queue (Getter) and leave it via the OUTPUT queue (Putter).

CRS controls which queues are available and how the item moves from one to another by gathering item information and run it through a set of rules.

CRS reevaluates the item after each individual processing step, computes item risk and routes it according to it.

Chapter 2

First steps

What do you need?

That depends on how you want to proceed. One way sample this book is to simply read through it. On the other hand, there is a lot more to learn if you try out things we discuss along the way. If you decide for the second approach, here are the ingredients that you'll need to use CRS:

- One computer with installed FraudOne.
Just make sure you can play around on it and will not annoy anybody when you create or delete data on it. Production systems are not recommended.
- A CRS license.
FraudOne comes in different flavors, not all of them including the CRS editor. Since you received this book, we'll assume you have that, since it is part of the CRS bundle.
- Getter file(s) with a couple of items in it.
You'll need some items to process. Don't take too many, since you'll just need to wait longer until they are processed. Something in the range of five to twenty should do fine. It helps if the items have matching accounts in the reference database.
- The necessary rights to start the Administration Client for rules editing and to run and control the system via the Server Monitor. You'll need to restart some components after changing the rules. (If you do not have FraudOne system access, you will need to partner with someone who does in order to work through the rules testing.)

Start the necessary system components

Assuming you have a working system, you'll need to start a couple of things. Grab your Server Monitor (or ask the system administrator – you will need to do this a couple of times) and start the following components in this order:

1. SignBase Servers (attach manager and server)

The SignBase Server incorporates the Configuration Server. There is no way you can change configuration items stored via the Configuration Server if it is not running.

i The Configuration Server is responsible for managing and distributing configuration information and to track changes to it. CRS uses it as the source for all settings and rules. You can find more information on it in the *Kofax FraudOne Administrator's Guide*.

2. Workflow Router

This will provide queue information. You should not need it if you start a new set of rules, but you need it for a running system. Since it uses the Configuration Server that has to be started first.

3. CRS Engine

The CRS engine is a prerequisite for components processing items. We will need it for ASV and SignCheck.

4. SignCheck Servers

The client will refer to it to get some information. SignCheck servers need a running Workflow Router, which in turn needs a running Configuration (SignBase) Server.

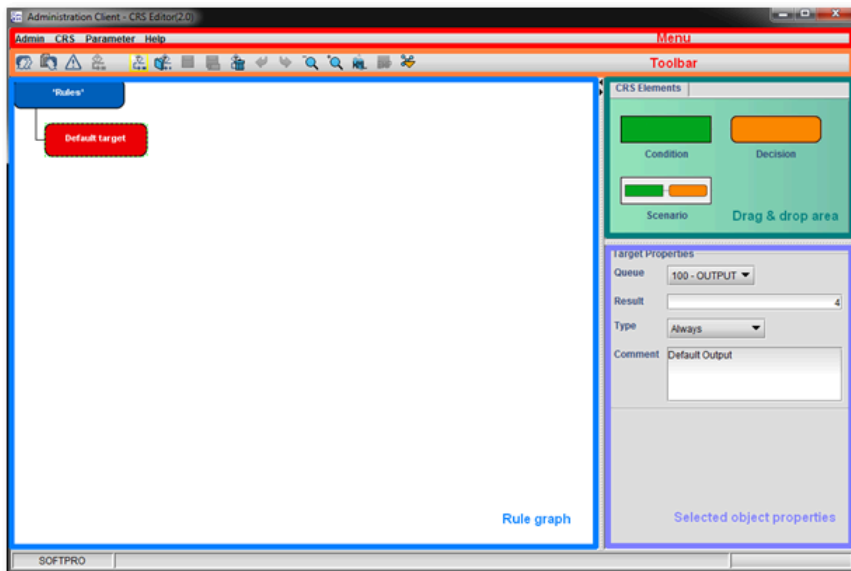
Now you are ready to go. You can now start the Administration Client.

Use the editor

Start by locating the Administration Client. You either have an icon on your desktop or run the procedure AdminClient.cmd from your FraudOne installation.

From the Administration Client main screen, you can call the CRS rules editor by selecting **CRS Editor** from the **Admin** menu or by clicking on the editor button.

Before you begin, here is an overview of the editor layout.



The main area you will be using is on the left. That's where the rule graph will be displayed. If you click on one of the graph objects, its properties will be displayed on the right. You can change settings here. On top of the screen, you will see the toolbar and the menu.

The most important feature is the yellow question mark, or the **Help** menu. This will give you more information on the rules editor.

To start the CRS editor, you will need to select **CRS Editor** from the **Admin** menu or click the according button on the toolbar.

Wrap up

CRS configuration is stored via the Configuration Server. It uses versioning and change tracking to let you find out which user changed what when.

This implies some system dependencies. The CRS engine depends on the Workflow Router, which in turn depend on the Configuration (SignBase) Server. No client, visual or automatic, will be able to run if Workflow Router and CRS engines are not started.

CRS configuration is changed via the CRS Editor in the Administration Client.

Chapter 3

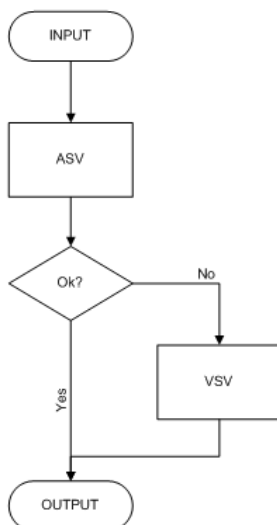
A simple automatic and visual signature workflow

What do you want to do?

Now that you are acquainted with the editor, start by configuring your first workflow. Start with something simple, such as verifying the signature on checks.

I hope you have some checks matching the accounts stored in the reference database. If not, you should consider creating some, or loading the necessary account information into the system you use for testing. Otherwise, all checks will be rejected. See the accompanying *Kofax FraudOne Administrator's Guide* or ask your administrator.

Here is my suggestion for the first simple workflow. In order to automatically verify signatures, you will need ASV (Automatic Signature Verification). The result of this step will determine what should happen to the document next. Items with a bad result need to be reviewed. Here is the diagram that represents this:



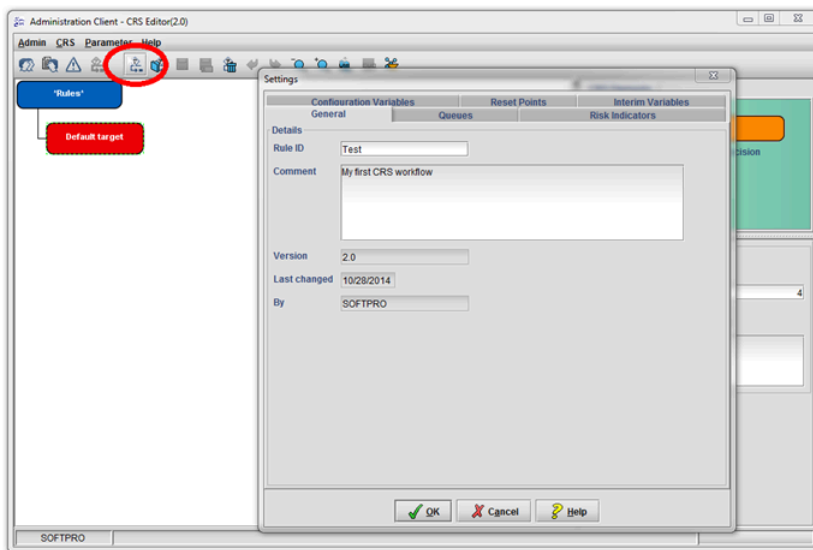
The Getter will load new items into the INPUT queue; from there you want to move them to ASV. If everything is ok (ASV return code 0), you will place them in the OUTPUT queue, to be accepted. If

ASV does not recognize the signature, you want to visually review the items. You will route them to VSV. The VSV user will then accept or reject the items.

Create a new rule set

The next step will be to write the CRS rules for the workflow above.

First, open the Administration Client and start the CRS editor. It will come up with an empty screen. Click the **Create new rule graph** button or select the menu entry:



A new rule set will be created and you will be prompted to enter some basic definitions.

Enter the rule ID. This will be used to identify the rule set from now on. Make sure you pick an ID that you haven't used before – it must be unique.

In the comment section you can specify some text that will help you remember what this rule set is meant to do.

The other fields are managed by the system. They tell us the version, last change date and the author of this rule set.

This information will also be displayed by the Workflow Router and CRS engine at start time. This helps you find out which rule set is in use.

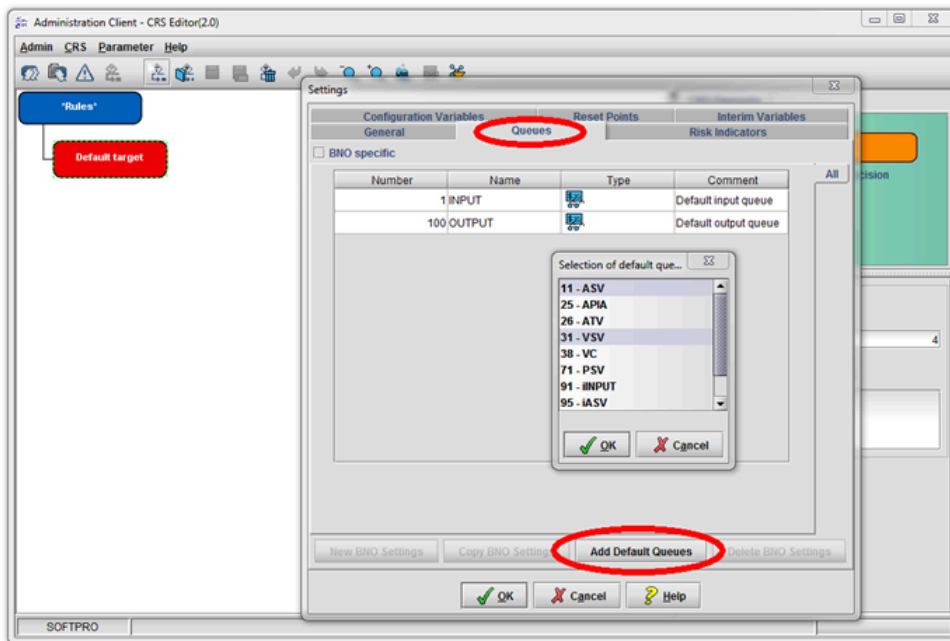
Before you can close this panel, you will need to define some basic configuration data:

- The queues you will be using.
- The risk indicators you will check.

Set up the queues

Now that you have a rule set, you need to specify the queues you are going to use. Look again at the workflow graph above. You'll notice that there are 4 queues: INPUT, ASV, VSV and OUTPUT. Here is how to set them up:

Select the **Queues** tab in the window. Notice that two of them are already in the list – you always need the INPUT and OUTPUT queues. Now add the others. Click the button labeled **Add default queues**.

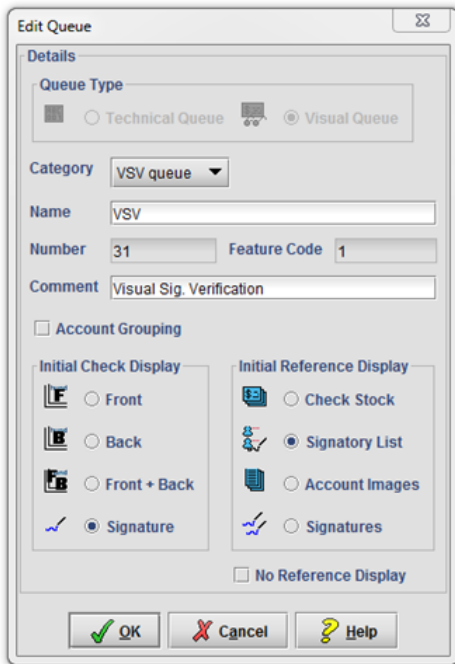


It will present you with a list of queues configured for your system. The content depends on the specifications you supplied when ordering FraudOne and can be changed in the Administration Client configuration.

You will select the missing queues ASV and VSV and add them by clicking the **OK** button. Your queue list should now be complete.

i If the setup you use doesn't provide a VSV queue in the list of default queues, right-click into the queue list and select **Add**. This will create a new queue for you. Set the queue type to **Visual queue** and the category to **VSV**. The rest should match the description above.

Before you leave it, check the VSV setup. The VSV queue is meant to be used for visual review. You may want to look at items that have been rejected by automatic signature verification. The Administration Client can be used to specify how the verification client should display items. Right-click on the VSV line of the listed queue table and select **Edit**. You can now change the queue properties:



On the left, you can specify how the item data should be shown. Select **Signature**, which means just the signature area, since you want to focus on the signature verification for this example. **Front** means the front side of the check, while **Back** points to the check back side.

On the right side you can specify how the Java Client should show the reference data. I propose you select **Signatory list** to do signature verification.

Next you will have to assign this process a feature code. This is the information used to store and identify user results for this queue. Set the VSV feature code to 31.

i It is recommended to use the same number for feature code and queue number. This makes identifying results a bit easier.

The queues are now done. Next come the risk indicators.

Choose risk indicators

Risk indicators are data you want to use in your decisions. This can be reference data, item (check) data, decisions made during item processing or external information.

Risk indicators can be:

- Built in:
Most of the commonly used risk indicators have been built in and are always available.
- Configurable:

Some installation dependent risk indicators need to be configured to be used. You will need to edit the DB helper configuration to make them available. See the *Kofax FraudOne Administrator's Guide* or to the example below.

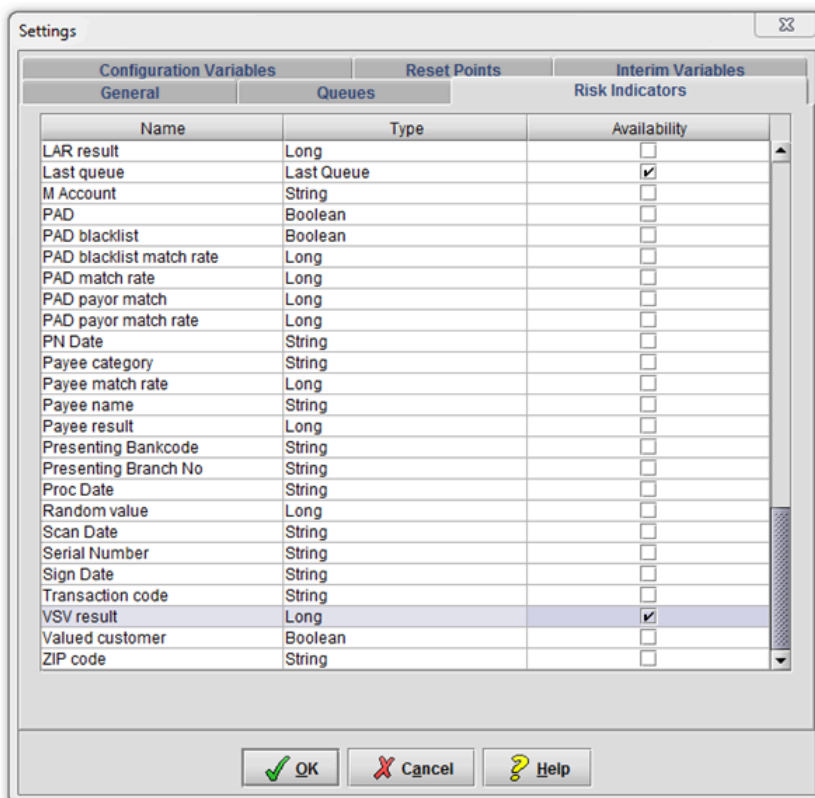
- Extensions:

If a customer installation uses non-standard data (new database tables created specifically for that customer), using that data as risk indicators requires Kofax to write a DB helper extension to make it available. This is not something the user can configure.

What risk indicators will you need? See workflow diagram. From the INPUT queue you want to move items to ASV. That means you will need to look at the **Last queue** the item was in. In ASV you want to accept items that have been processed correctly and move items to VSV that have been rejected by ASV. That means you will need to look at the **ASV result**. All items from VSV will be moved to OUTPUT, but you want to set the final result depending on the VSV user decision. You will need the **VSV result** for that.

The next step is the risk indicators. Select the **Risk indicators** tab. You will be presented with a list of risk indicators available on your system. This includes built in indicators, configurable indicators and extensions.

Select the indicators named above:



Indicators that you did not select will not be available for making decisions. You can always add other indicators later. It is important you only select the risk indicators that you are going to use. One reason is, that the drop-down lists in the editor will be cluttered with data you don't need. The

main reason is that the system will load the data for all selected indicators each time it needs to make a decision. Having non-used indicators selected will decrease system performance.

If some of the risk indicators don't show in the list above, you might need to configure them. See chapter [Defining your own risk indicators](#).

i In some versions **VSV result** might show as simply **VSV**. You can use that.

Default target

You can now close the settings window in the CRS editor.

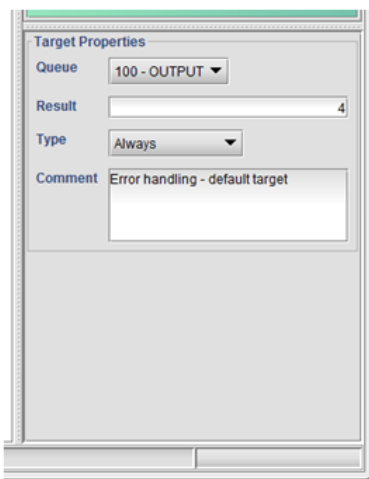
Have a look at the rules graph on the left side. You will see two objects: one is named **Test** (or whatever name you have given your rule set), while the other is named ***Target***. The '*' around it indicates you need to edit this object before you are allowed to save it.

But what is that 'Target'? It is a default target used for error handling. You cannot delete it. When making a decision, CRS will parse your rules and do whatever the decision element at the end of a rule tells it to do. However, if no rule applies, it will end up in the default target.

You can use the default target to specify an action if no rule covers the case. Use it wisely, just for error cases. You will know you need to revise your CRS rules if the default target has been used!

The default behavior is to put items into the only safe place to go that exists in any setup – the OUTPUT queue, with an error return code that will tell others the item has not been processed. If you wish, you can design another mechanism in your system – you could route those items to a visual review queue where a user can manually decide them. Just be sure you don't make any errors when writing the rules for that queue, or else you'll end up hitting the default target again.

For this example, set the default target to point to the OUTPUT queue, use a final result of 4 (not processed) and hope the CRS will never have to use it.

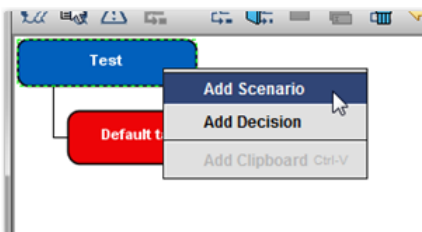


It is very important to set your default target correctly. That's why the editor will not let you save the set if you did not at least look at it. Notice that the '*' indicators go away after you selected the object by clicking on it and applied the changes.

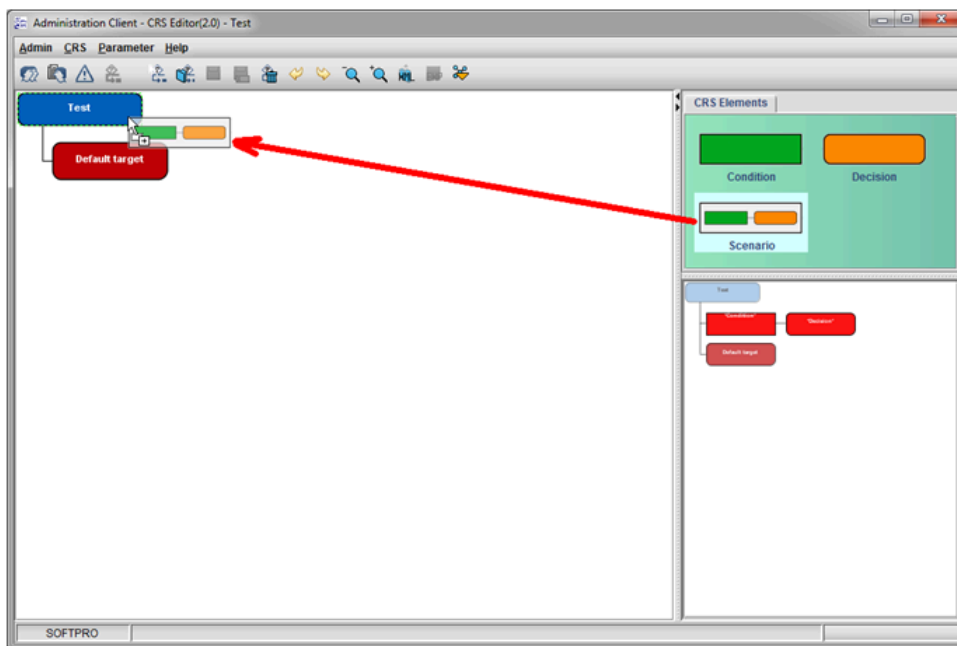
Your first condition

You now have a rule set with defined queues, risk indicators and error handling. It just does not do anything yet. It is time to define the first rules.

Right-click on the top object (labeled 'Test' in the sample case) and select 'add scenario'. You will see something like this:



Alternatively, you can also drag a scenario over from the drag area and drop it on the top object:



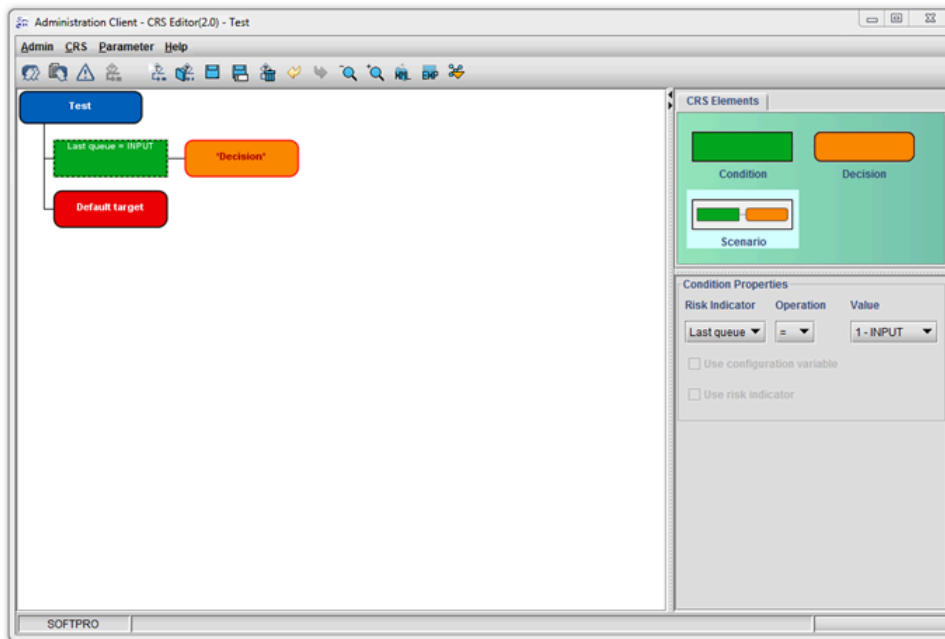
This is the first rule. The green block on the left (colors can be configured) is called a condition. The rule ends in a decision (last block on the right).

CRS will parse the rules from top to bottom and will walk down a branch if and only if all conditions along the path are true. If it ends with a decision bloc, the decision is carried out and the rule

parsing ends here. If a condition is false, the parsing continues with the next condition in the tree. If none of the rules passes, the default decision will handle the error.

Backing up a little think about what you want to tell CRS to do? You want to move every item from the INPUT queue to ASV. That should not be too hard:

Select the condition block by clicking on it. On the right side of the screen, you will now see its parameters. Click on the risk indicator drop-down list and select **Last queue**. Select = as operation and select **1 - INPUT** as value. The condition now says "I'm true if the last queue the item was in was INPUT". Only items coming from the INPUT queue will pass this condition.



Other conditions based on other risk indicators will be edited the same way. Only one risk indicator can be used per condition. Operations available depend on the indicator types used. Integers and floating point indicators support less than, greater than, equal and ranges, while strings can be validated using regular expressions.

Make a decision

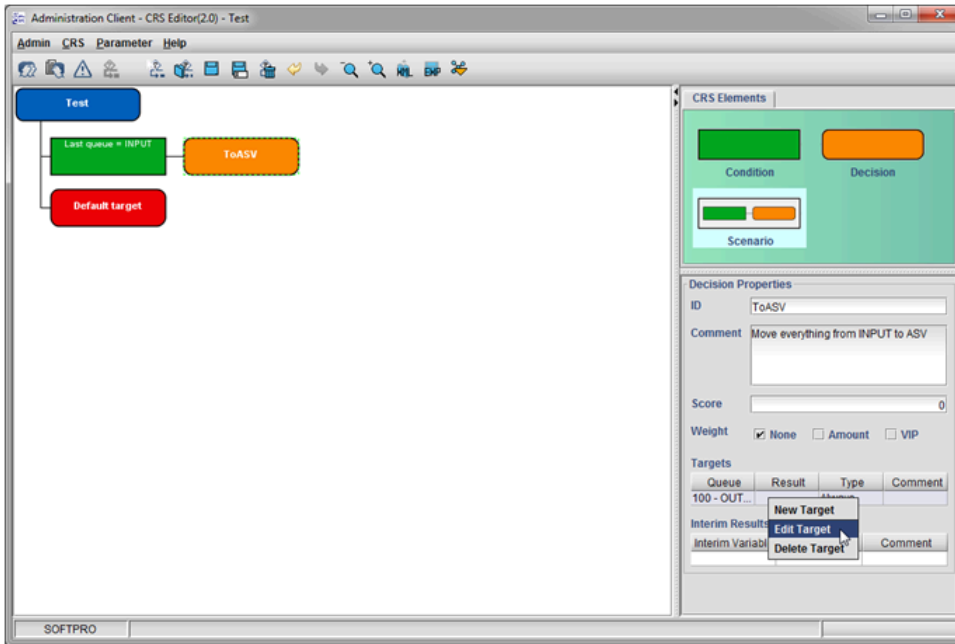
You now have the condition that selects items coming from the INPUT queue. You will need to tell CRS what to do with them.

Select the decision object on the right.

The **ID** parameter will be used to identify it. You will give it a more meaningful name like **ToASV**. Even though the system does not enforce unique decision names, it is good practice to use them. Rule traces will contain the decision name and you'll not be able to tell them apart if they have the same name.

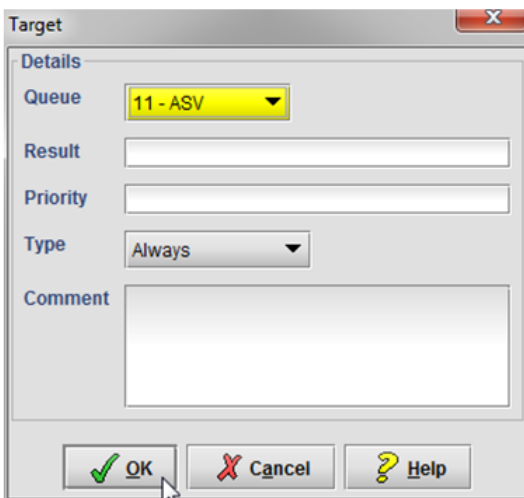
The **Comment** is not mandatory, but can be used to store some information about this decision (rule).

Score and **Weight** are used to calculate item risk and priority. Those will be discussed later.

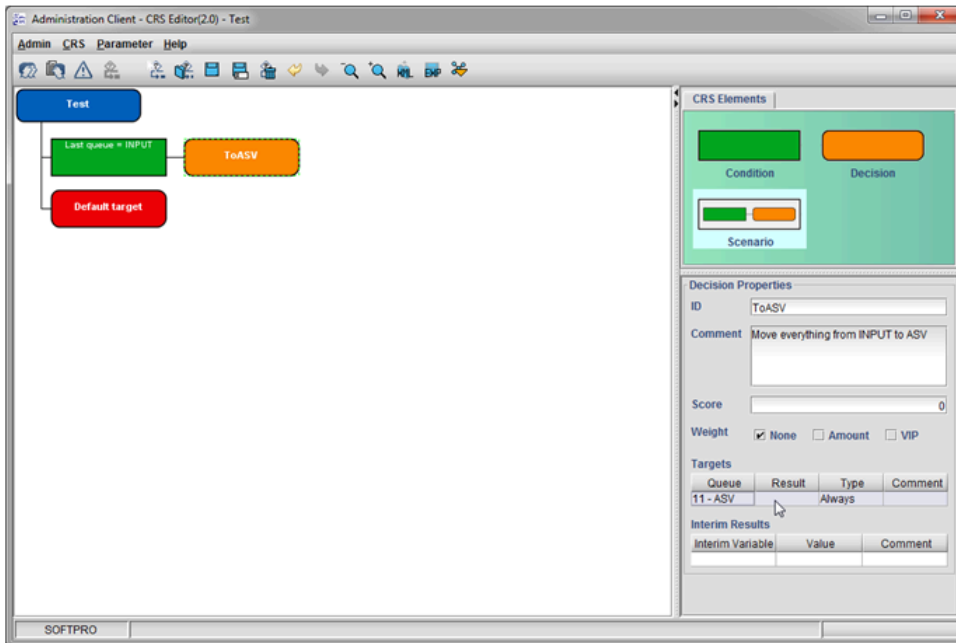


What we now want to do is tell the items where to go. Right-click on the existing line in the list of targets and select **Edit target**.

The items should go to the ASV queue. From the **Queue** list select **ASV**. You can leave **Result**, **Priority** and **Comment** empty. Set the routing type to **Always**.



The decision object will now look like this:



CRS will parse the rules tree. If it reaches this decision object (it does so if the last queue is INPUT) it will proceed by checking all targets contained within. Yours only has one, and the type **Always** always applies. Thus, the item is moved to this target, which is the ASV queue.

Complete the rules

You can now write the rules for the rest of the workflow graph.

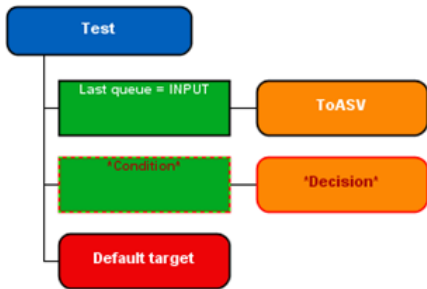
You have just done the INPUT to ASV routing. The next determination is to decide what to do with the items coming from ASV.

ASV routing

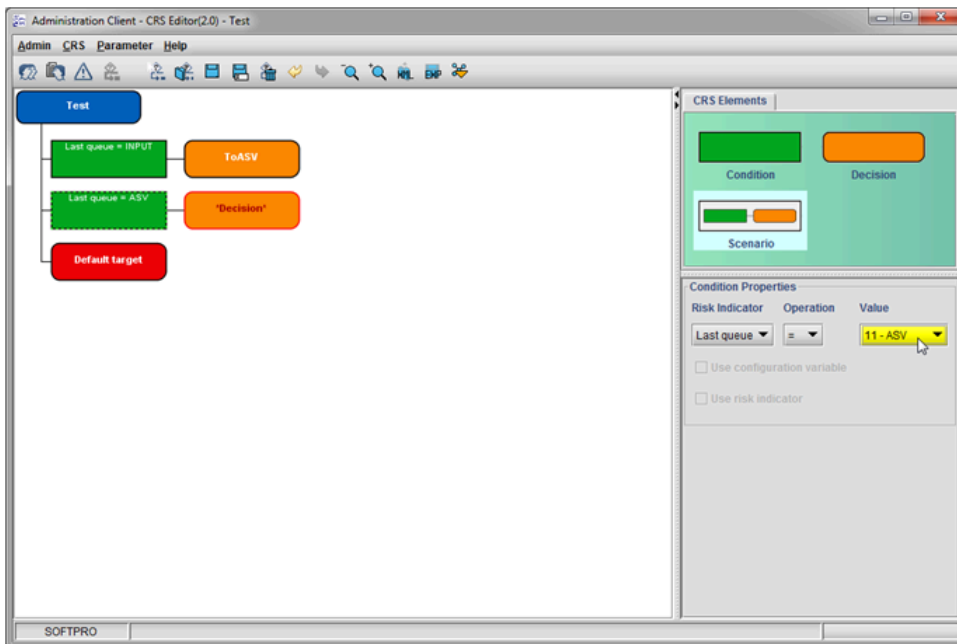
Take another look at the workflow graph. If the ASV result is ok (0 – item passes review), you will want to accept the item. This means to route it to OUTPUT with a final result of 0.

If the ASV result is non-zero, you will want to move it to VSV, where a user can visually review the item.

Go back to the CRS editor, right-click on the top object (labeled **Test** in this case) and select **Add scenario**. A new scenario (rule) will show up. It does not contain any data yet.

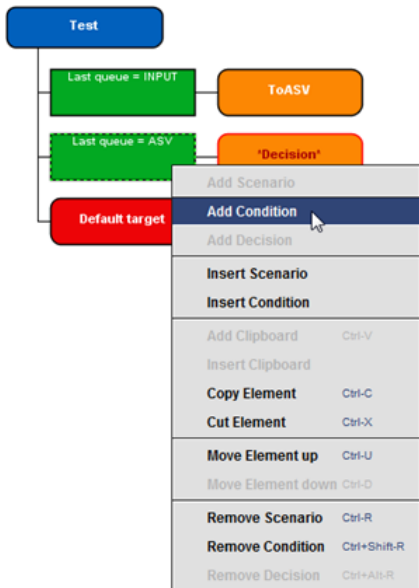


Right-click on the new condition. Remember, this rule is designed to work on ASV items. You will use the first condition for this. Select the empty condition and set the risk indicator to **Last queue** and the value to **11 - ASV**.

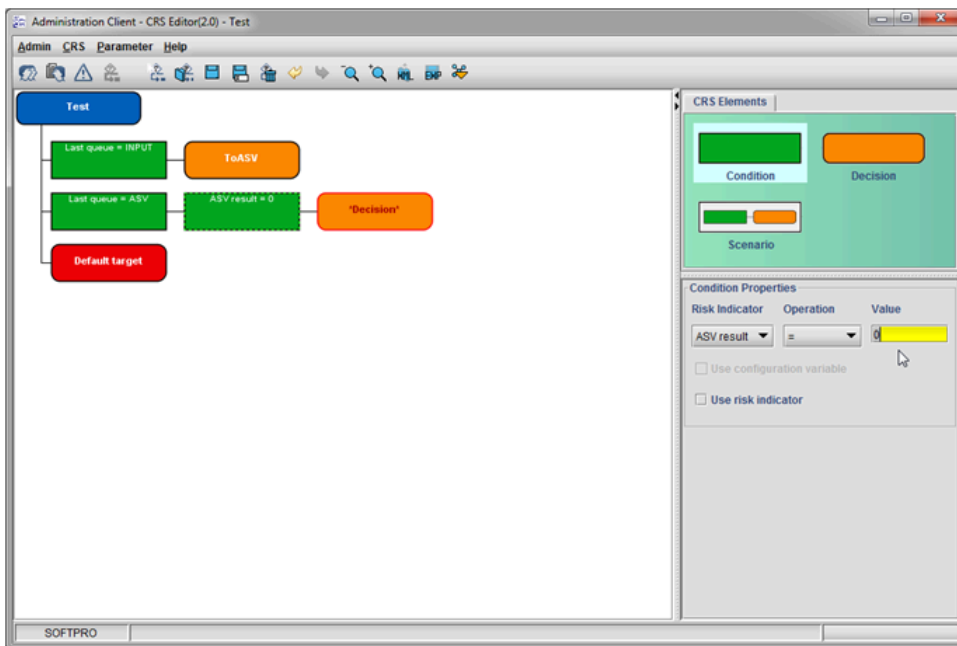


Now the remainder of this rule will only apply to ASV items.

The next thing you want to check is the ASV result. You will need another condition for this. Right-click on the first condition and select **Add condition**. Another condition will be appended after the one you selected.

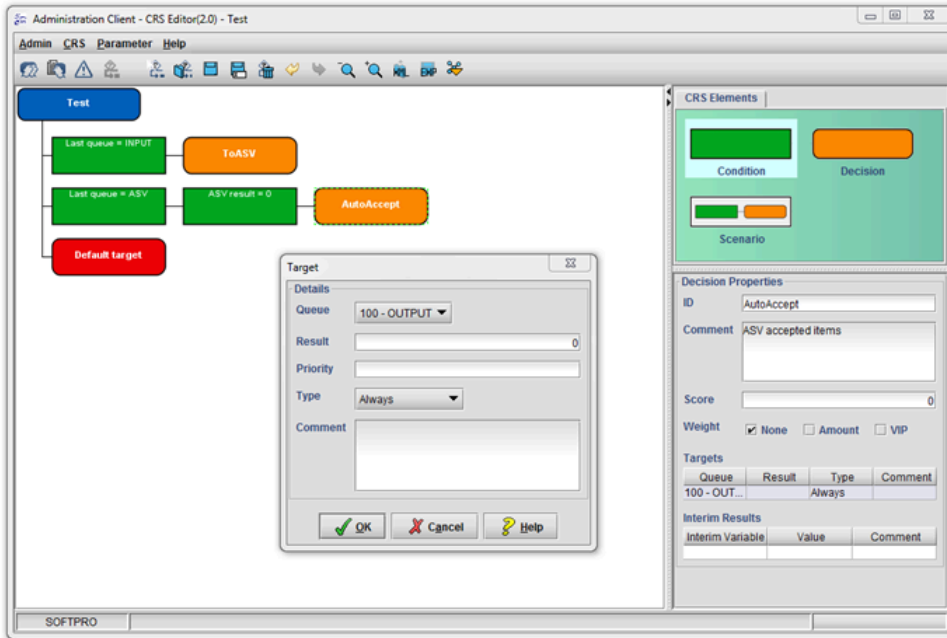


Click on the new condition and edit the parameters. Set the risk indicator used to **ASV result**, the operation to = and the value to **0**.



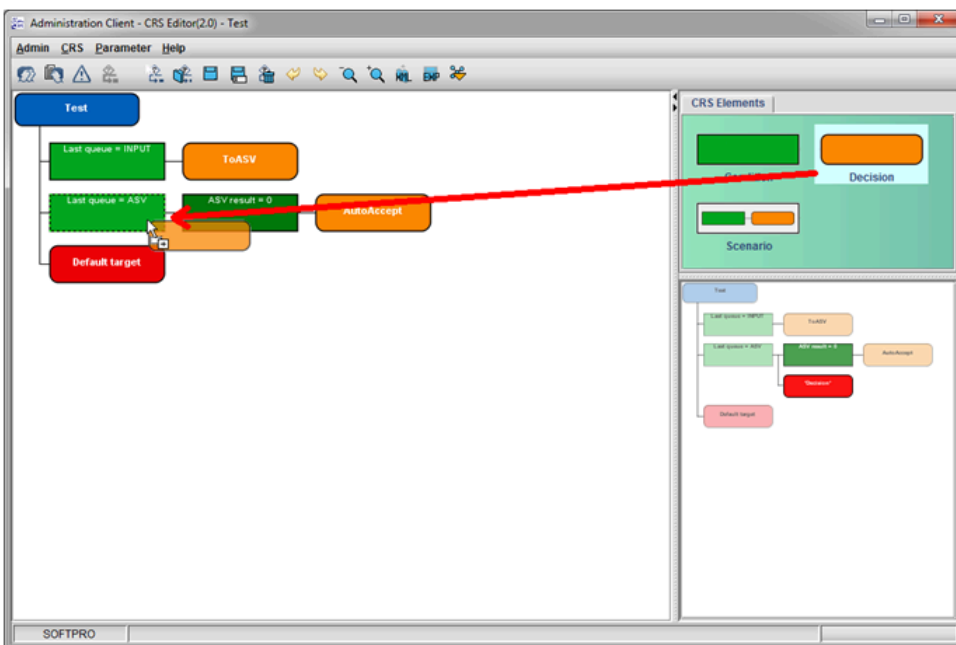
This condition will now only pass for items that have a zero (ok) ASV result. These items should be moved to the OUTPUT queue.

Click on the decision object at the end of the rule. You will name it **AutoAccept**. Edit the target in the list by right-clicking on it and selecting **Edit**. The queue will be **100 - OUTPUT** and the final result you want to set **0**.

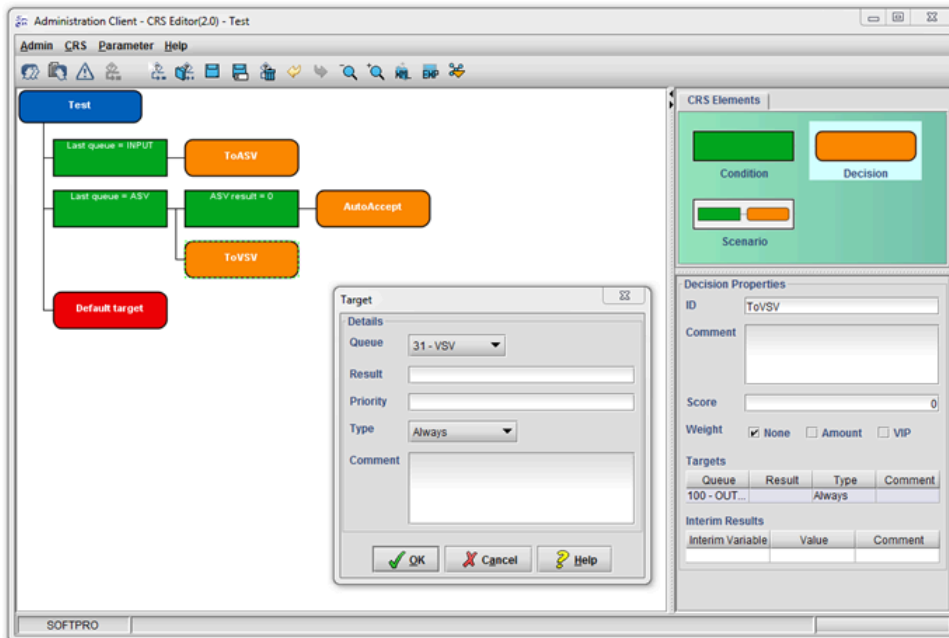


This will move items that have been accepted by ASV to the OUTPUT queue with a final 'ok' (result = 0). You now only need to take care of the items that have been rejected by the automatic signature verification.

You will do this by right-clicking on the first condition (**Last queue = ASV**) and adding a new decision (or by dragging a new decision onto the **Last queue = ASV** condition):



A new decision will be added after the **Last queue = ASV** condition, below the **ASV result = 0** condition. Click on it and change the properties. Name it **ToVSV** and change the target in the list to queue = '31 - VSV'.



What will CRS do with ASV items? It will move along the rule graph until it hits the **Last queue = ASV** condition. After passing it, it will try the next condition, which reads **ASV result = 0**. If this is true, the condition will be passed and the item will end in the **AutoAccept** decision that moves it to OUTPUT. If the item has been rejected (ASV result is not zero), the **ASV result = 0** condition can't be passed and CRS will continue with the next object down the graph, which is the **ToVSV** decision. This will move the item to the **VSV** queue.

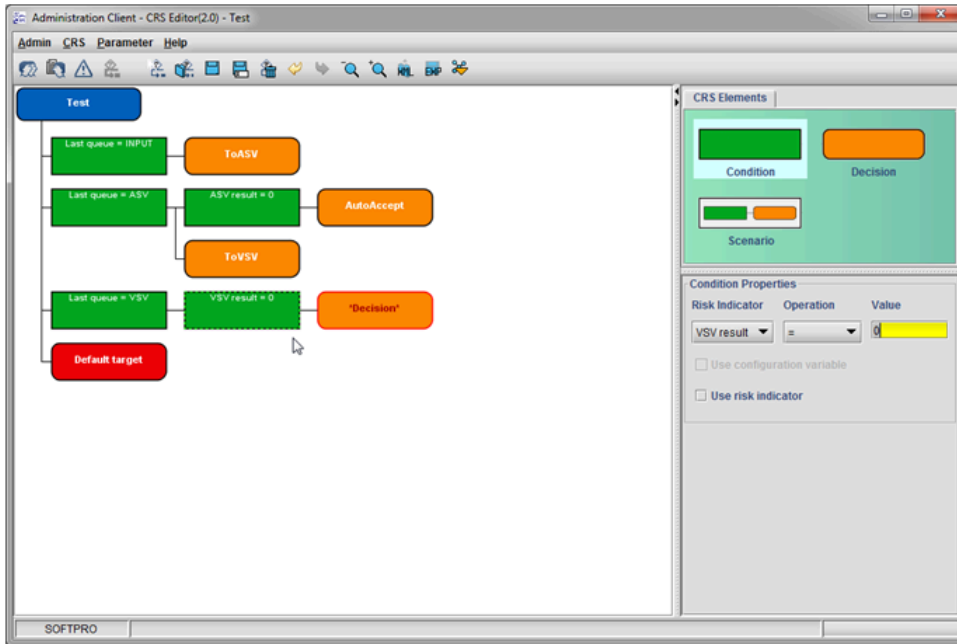
VSV routing

Now that ASV has been taken care of, you will need to tackle VSV routing. Obviously, you'll start with a **Last queue = VSV** condition. You should now be able to add that on your own.

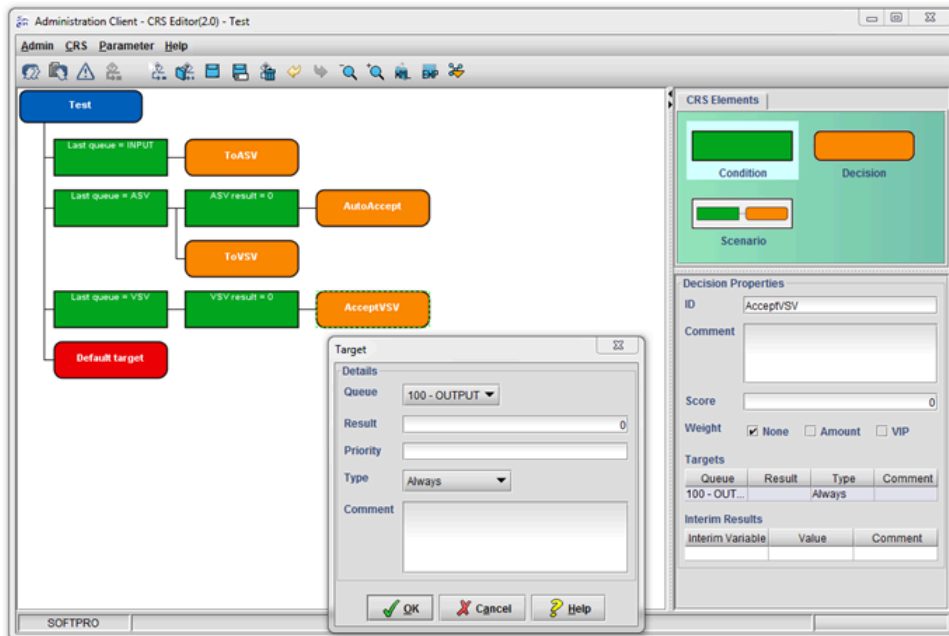
i Right-click on the top object **Test** and select **Add scenario**. Then click on the condition of the new scenario and edit the fields. Set the risk indicator to **Last queue** and the value to **31 - VSV**.

This scenario, or rule, now applies to VSV items only. You could now set the decision to point to the OUTPUT queue and be finished with it. But there is one more thing you might want to do - to set the final FraudOne result for the items. It should be 0 for accept and 1 for reject. Zero is fine, since the VSV result will be the same, but all the non-zero VSV results (range 01 to 99) would need to be converted to 1.

You need another condition to do that. Add a condition after the **Last queue = VSV** and edit it to read **VSV result = 0**:



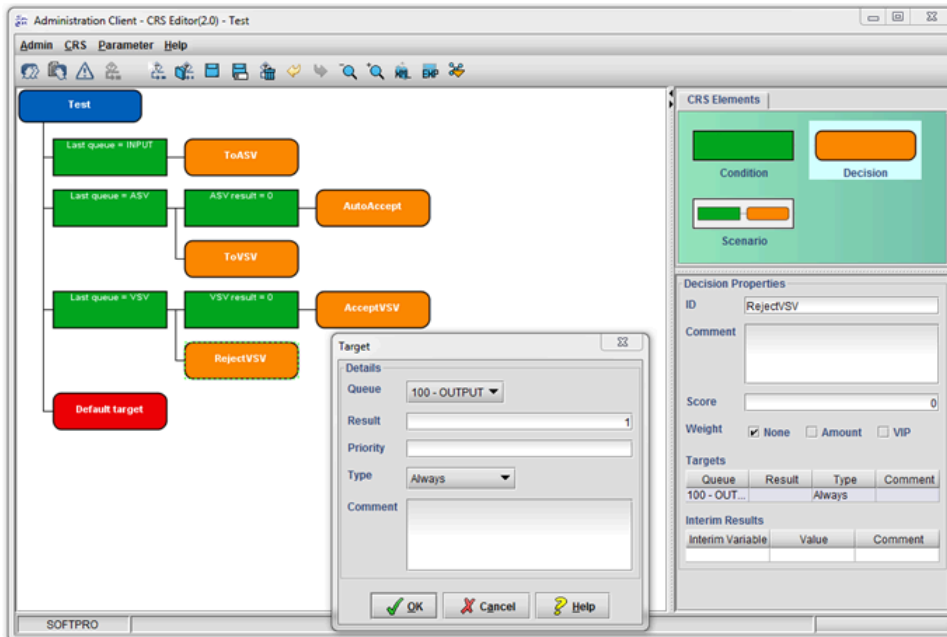
The decision behind it will apply to all items accepted by the reviewer in VSV. You will route the items to OUTPUT with a zero return code.



Now the only things to take care of are the items rejected in VSV (VSV result non-zero). You'll add another decision after **Last queue = VSV** and below **VSV result = 0**.

i Right-click on the **Last queue = VSV** condition and select **Add decision**.

Click on the new decision, set the name to **RejectVSV**, and set the one target in the list to point to OUTPUT with a new result of 1.

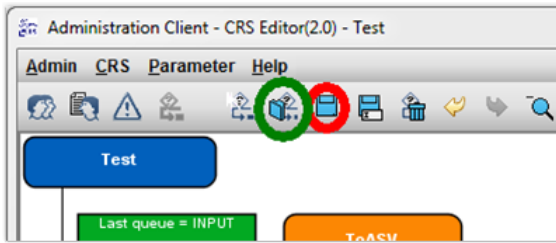


As you just found out, the targets in the list tell the decision object where an item needs to be moved to. In most cases, one target will do. The others will be handled in the next chapter. The most important item in each target is the queue field. It is mandatory and specifies the destination of the item.

Result, **Priority** and **Comment** can be left empty. The system will use defaults if none are specified. The default for the new result (it is only valid until the processing result in the new queue is set) is the old one. The **Priority** and **Comment** fields will be computed (see chapter [Score, risk and priorities](#)). However, you can override the values by specifying them in the **Result**, **Priority** or **Comment** field. You have done that for the result, since you want to change all non-zero VSV return codes to read 1.

Save it and choose the active rule set

Save the rules set by clicking **Save** on the toolbar (red circle below):



The rule set is now saved on the configuration server.

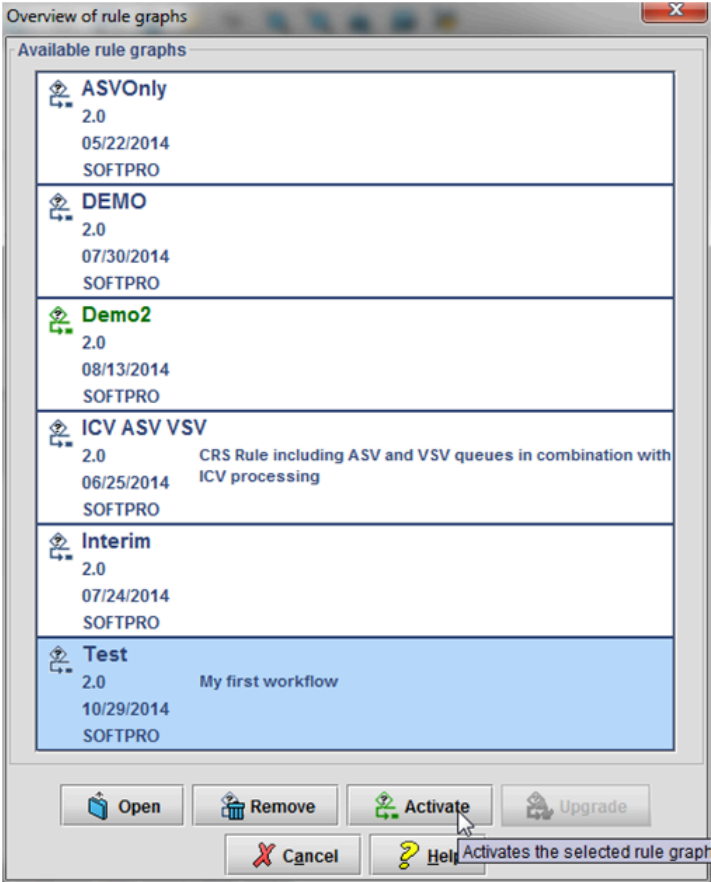
Which rule set is going to be used is determined by two settings:

1. The CRS XML file setting of the Workflow Router and CRS engines.
2. The mapping performed by the CRS editor.

The input file name can be set up for the engines and router. It will default to crs.xml but can be changed to a different name, such as Test.xml. This would need to be done for the Workflow Router and all CRS engines.

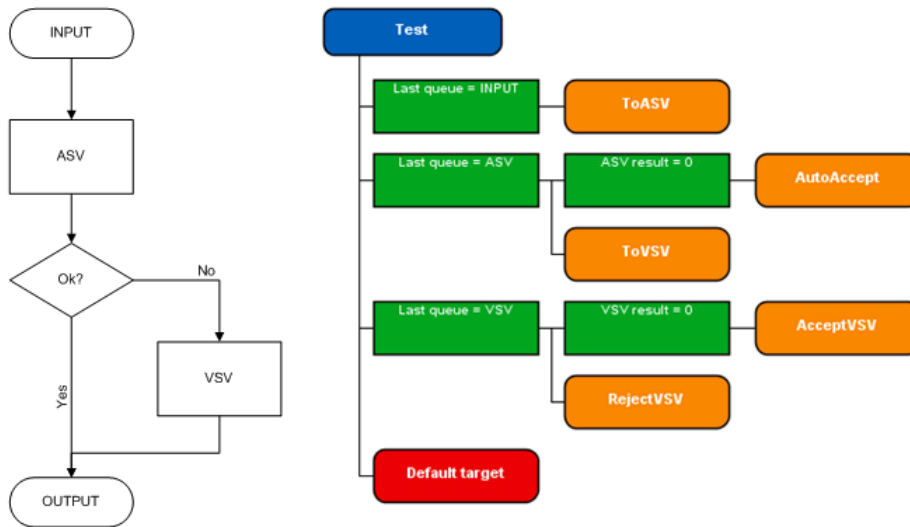
To make this a bit easier, the CRS editor can map one of the saved rule sets to the name crs.xml. If you leave the engine setting unchanged, you could thus activate one rule set or another, by mapping it to the crs.xml name.

If you have more than one rule set stored, you can click the **Set active rule graph** button in the toolbar (green circle in the image above). A list of all available rule sets will show and you can pick the active one:



Wrapping it up

The rule set up is now complete. Compare the workflow graph and the CRS rules for it:



Items start in the INPUT queue. Moving down from top to bottom, they'll hit the **Last queue = INPUT** condition, pass it and be sent to ASV.

After getting an ASV result, the same thing will happen again. Now they will not pass the **Last queue = INPUT** condition since that's ASV, but move downward, until they hit the **Last queue = ASV** condition. This is going to pass for items coming from ASV. The **ASV result = 0** will split that stream in two. Good items will pass it and hit the **AutoAccept** decision, while items that failed ASV will not pass it but fall into the **ToVSV** decision.

The VSV result will be processed by the branch starting with **Last queue = VSV**. All will be moved to OUTPUT, with different results. Accepts will hit **AcceptVSV** and get a 0 result, while rejected items will hit **RejectVSV** and get a 1 result.

And that's about it.

If you want, you can now test your rules. Assuming you saved the set and activated it, the only thing left to do is restart the servers. Stop everything and start all components. You will need:

- SignBase Servers
- Workflow Router
- At least one CRS engine
- SignCheck Servers
- ASV Servers
- ASV Clients

You can then start the Getter and load the items, watch how ASV handles them, process them with the visual verification Java Client and let the Putter run to generate the result file.

If you use the Java Client to access VSV, make sure the user ID you are using has review rights on the VSV queue. You might need to use the Administration Client to change the user rights and enable VSV review for the id before you are able to use it.

If you want to see what happens to the items, set the CRS engine trace to level **INFO** using the Server Monitor. The CRS will show which rule applies and what happens to the item, together with some timing information.

If you want to be prepared for the next run, clean the system by running DFP (day's final processing).

Wrap up

You can change the rules by adding condition and decision elements to your rule graph.

CRS parses the rule graph from top to bottom, left to right. It will pass a condition element if the condition is true. If not, it will continue down the next path available.

Parsing will stop when the process hits the first decision. The actions specified in the target list of that decision element will be carried out.

This process happens each time the item receives a new result (after completing processing in each queue).

The item will leave the FraudOne workflow when it reaches the OUTPUT queue.

CRS does not only control how items are moved around, but also which queues are available, how those queues are displayed in visual clients and which risk indicators will be used in this process.

Chapter 4

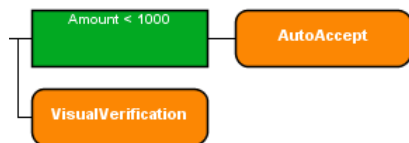
Rule blocks

When do you need rule blocks?

Complex rule graphs can contain repeating logical patterns in different branches. Rule blocks allows to group elements of such patterns together, give a name to the group and re-use the group further on in the rule graph as a single element.

Let's look at the following example. After several automated verification steps the level of verification can be considered to be sufficient for checks with small amounts but checks with bigger amounts should pass an additional visual verification.

The corresponding workflow pattern looks like this:



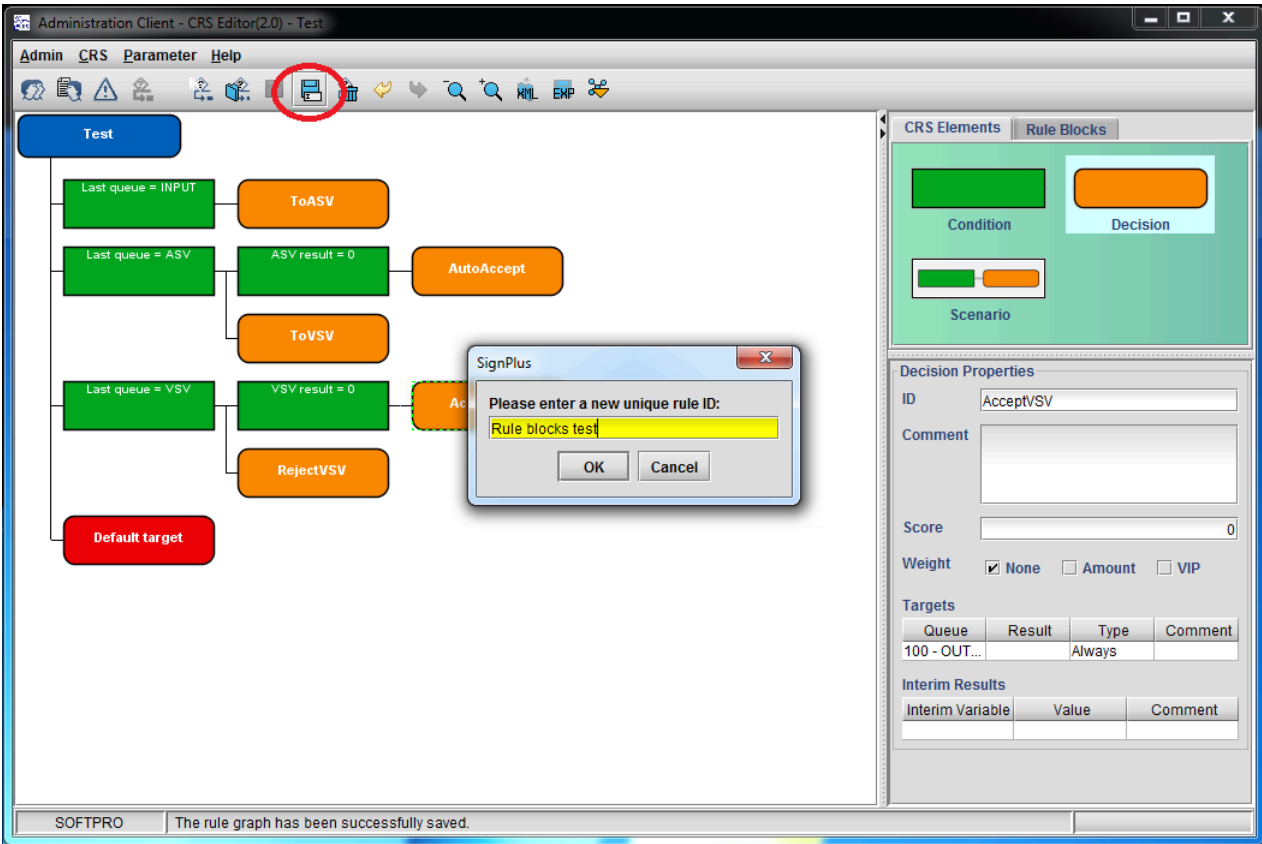
It is possible that there are more locations in the rule graph where this pattern applies. So, the rule graph could contain a number of identical elements like the pattern above in every such location. Now using a rule block

- is less error prone
- is faster to enter
- results in a more readable rule graph

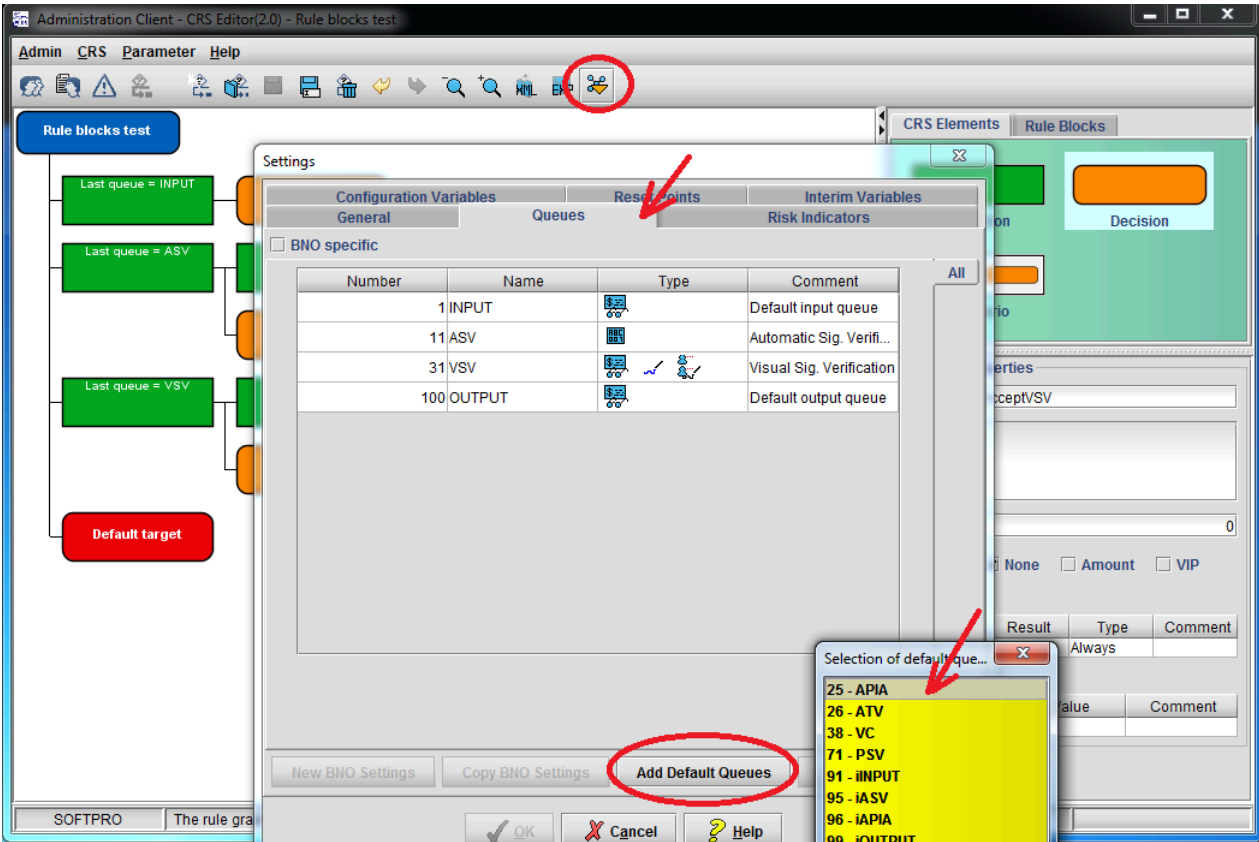
Let's give it a try.

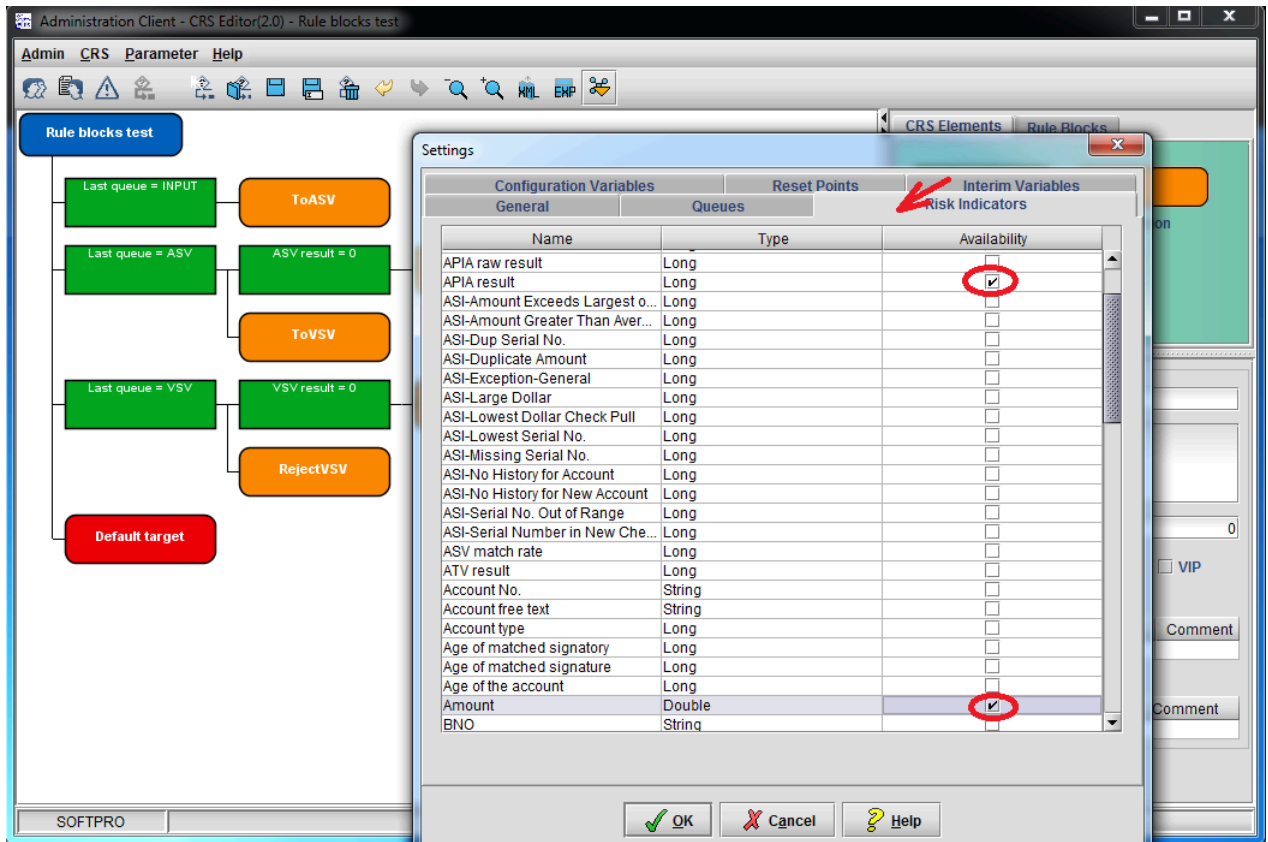
Create a rule block

Let's keep current rule graph for further chapters and create a copy of current rule graph under a new name:

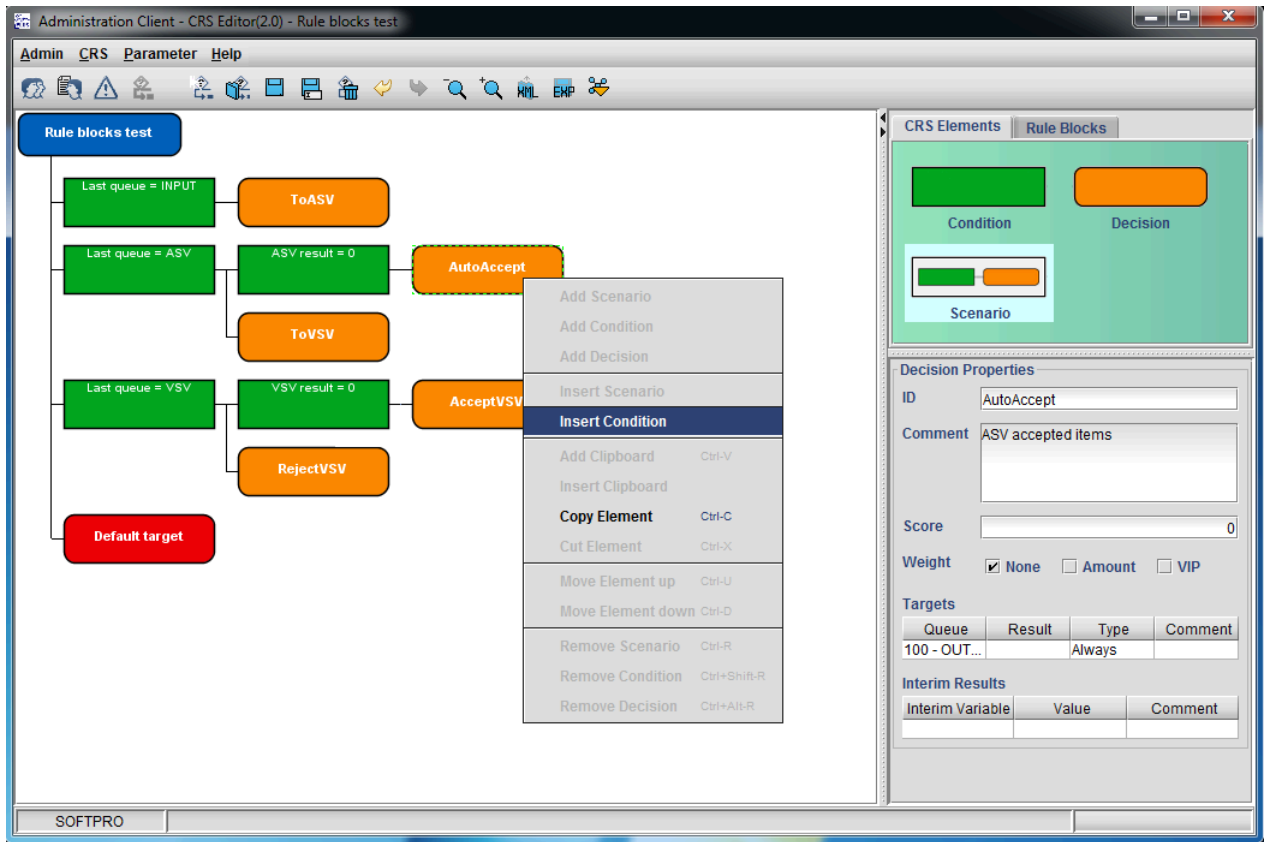


We will need an additional queue **APIA** and two additional risk indicators **APIA result** and **Amount**:

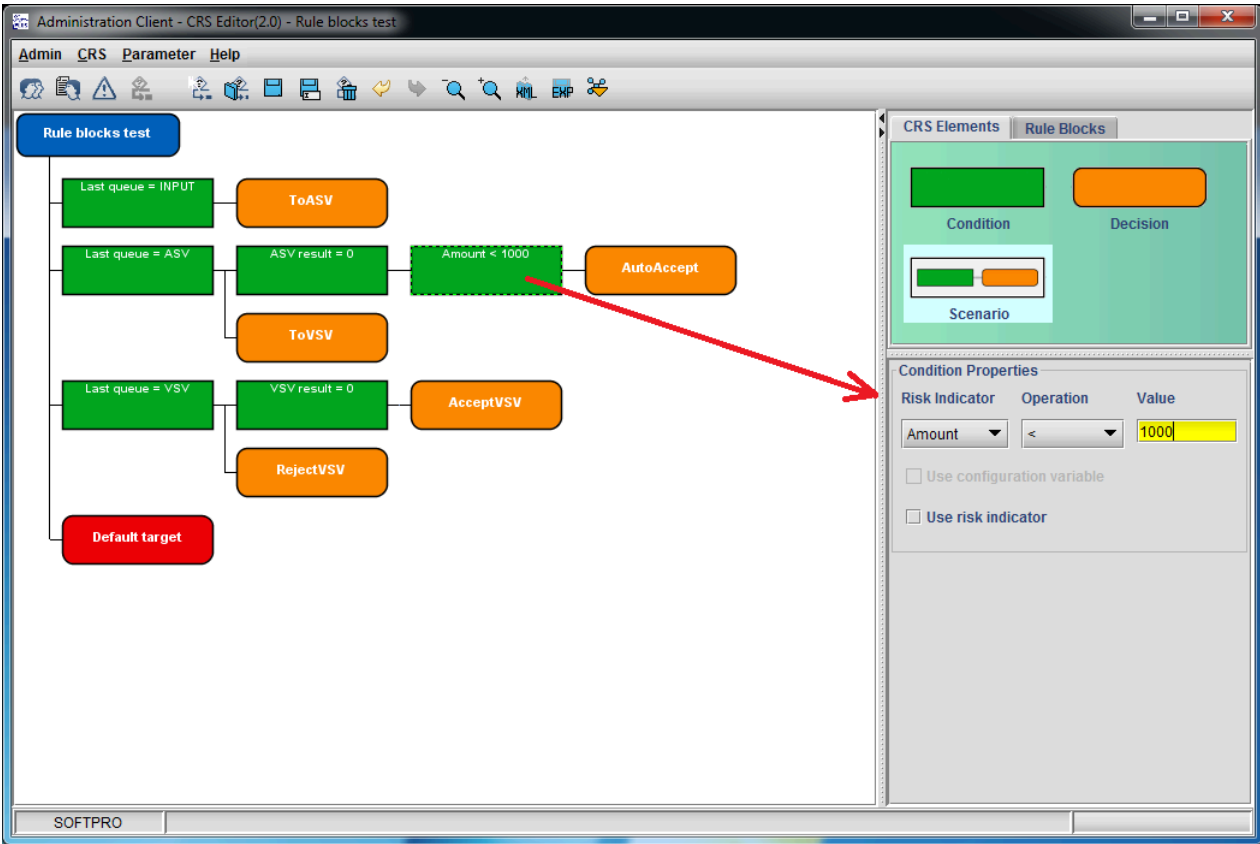




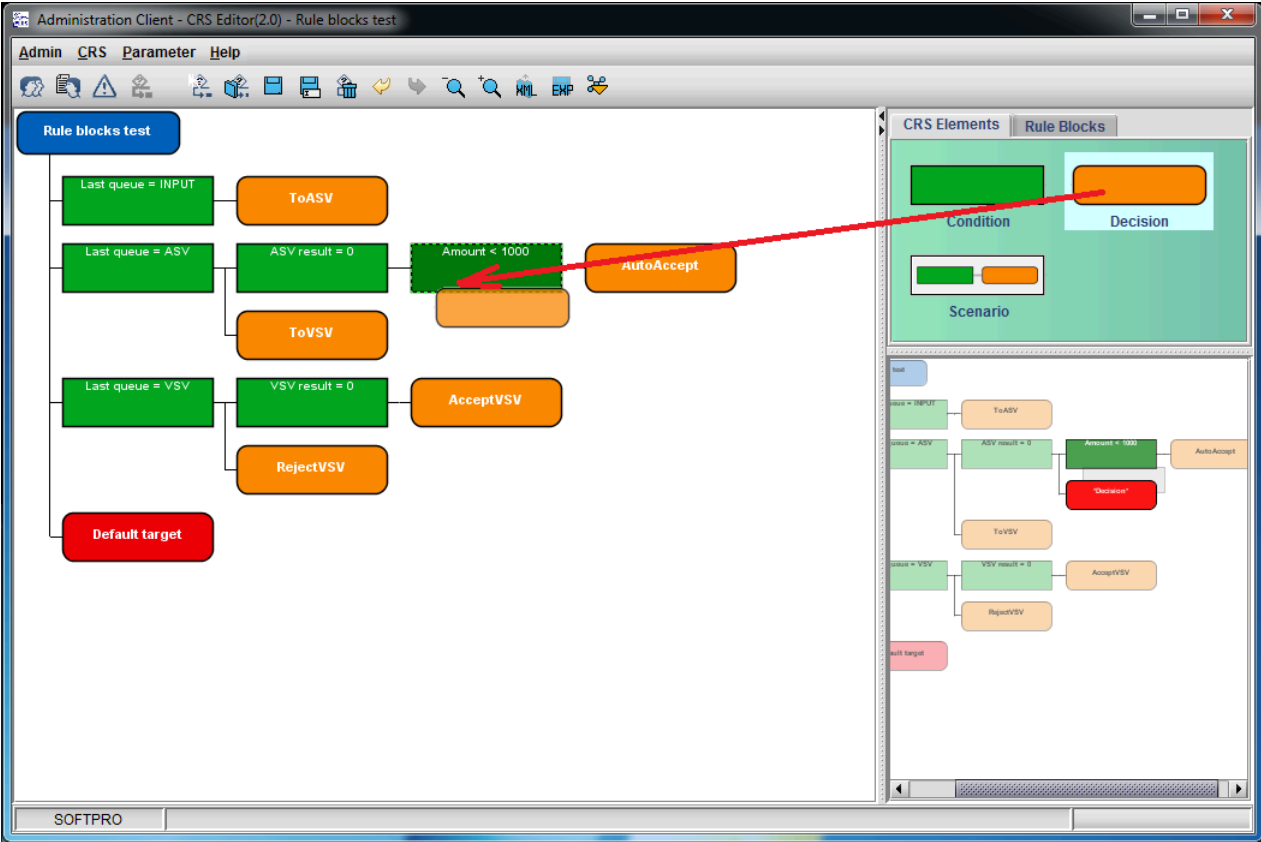
Now in order to check the amount let's insert a new condition before the decision called **AutoAccept**:



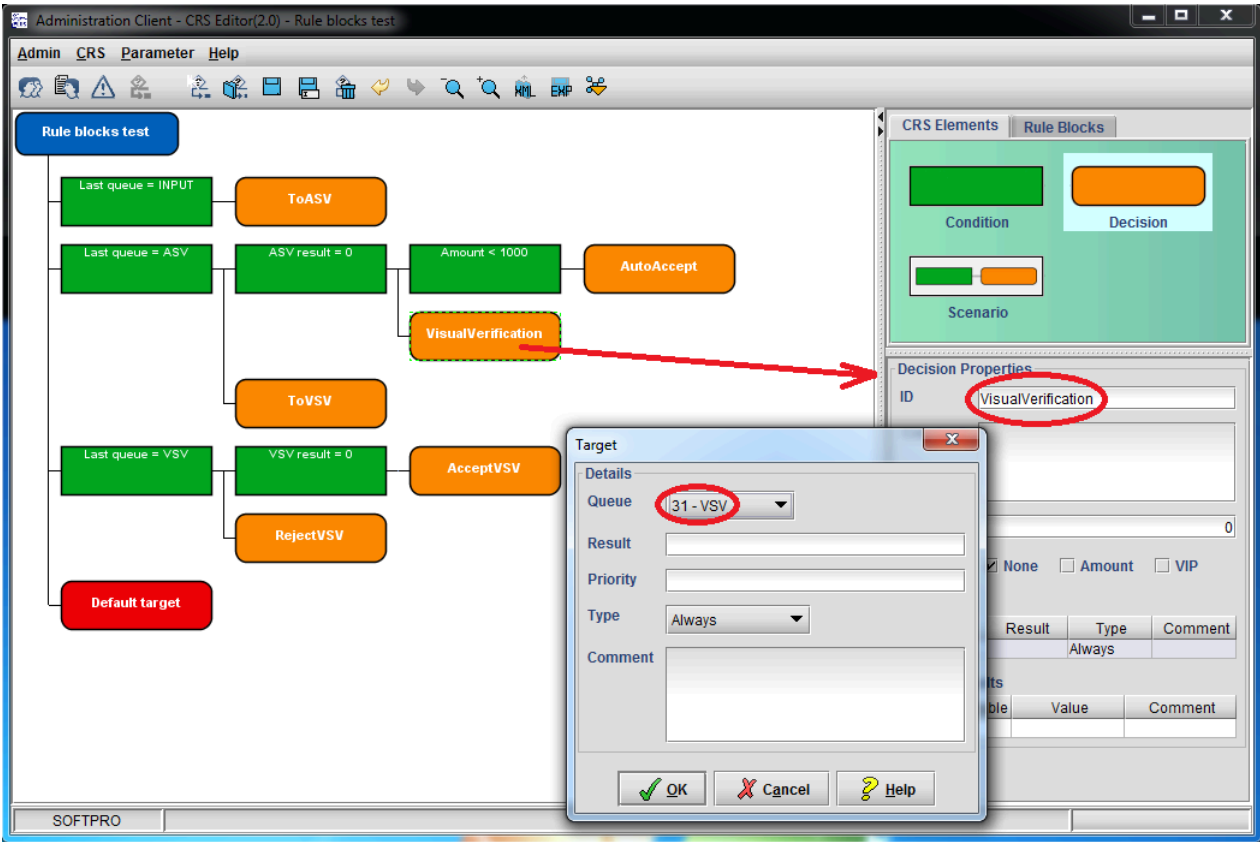
In the properties of the new condition, we set the **Risk Indicator** to **Amount**, the **Operation** to < and the **Value** to **1000**:



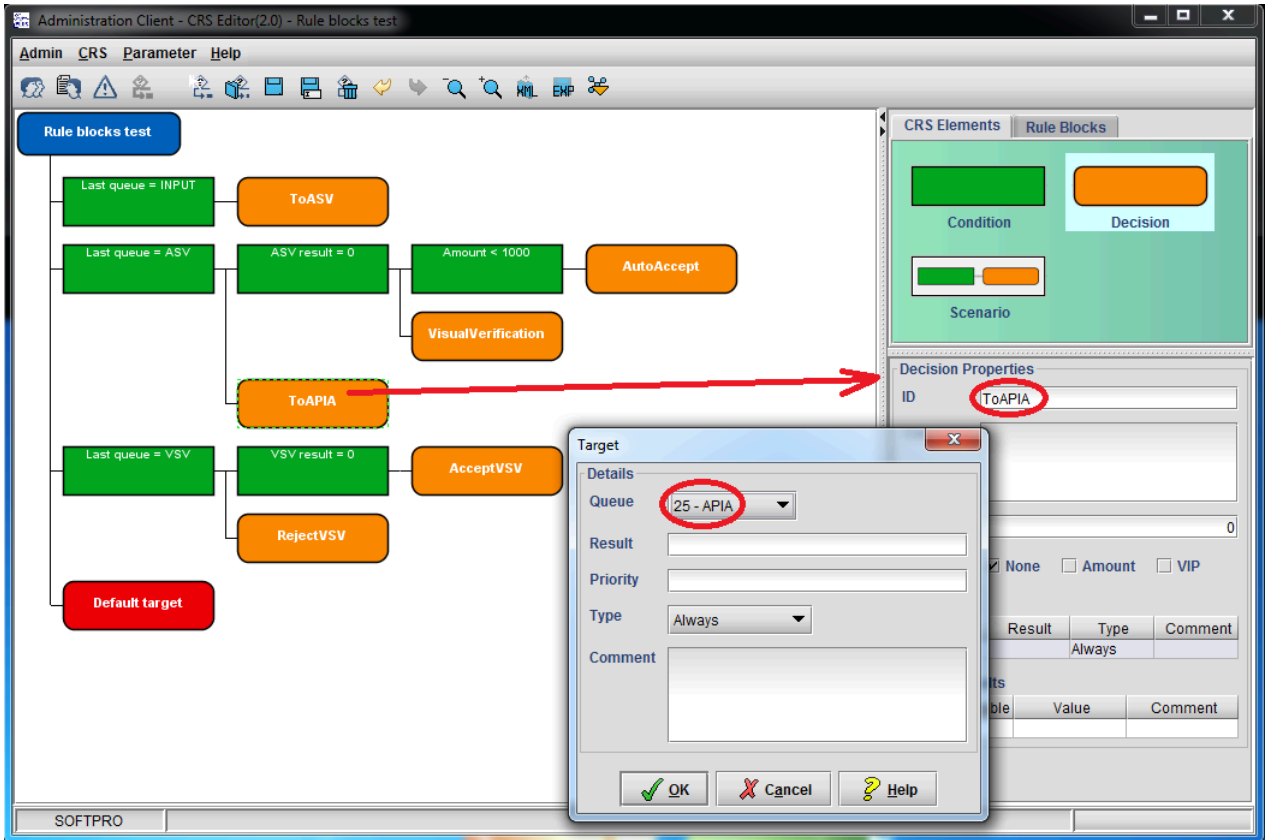
Then drag decision out of CRS elements panel and drop it just below the new condition:



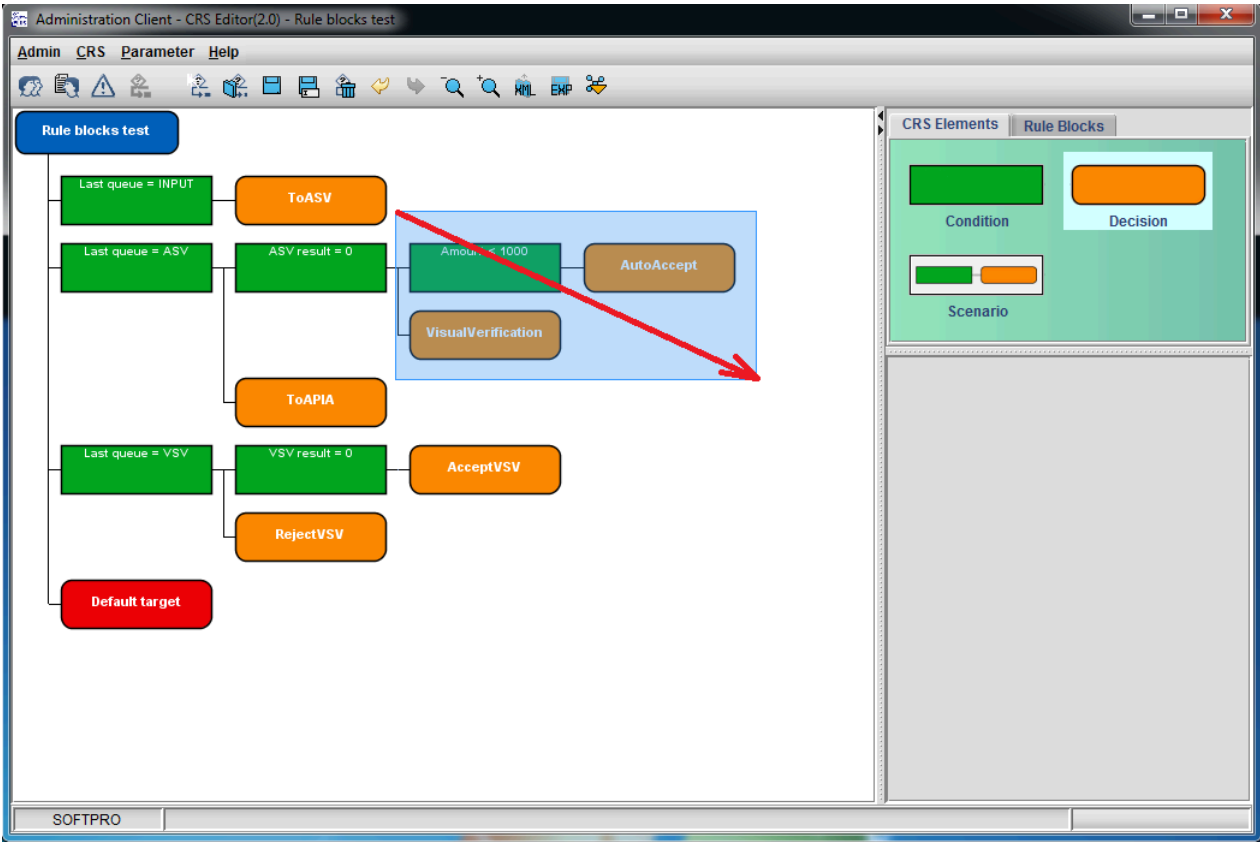
Here we want to send items with amount bigger than 1000 to visual verification. Let's configure the decision:

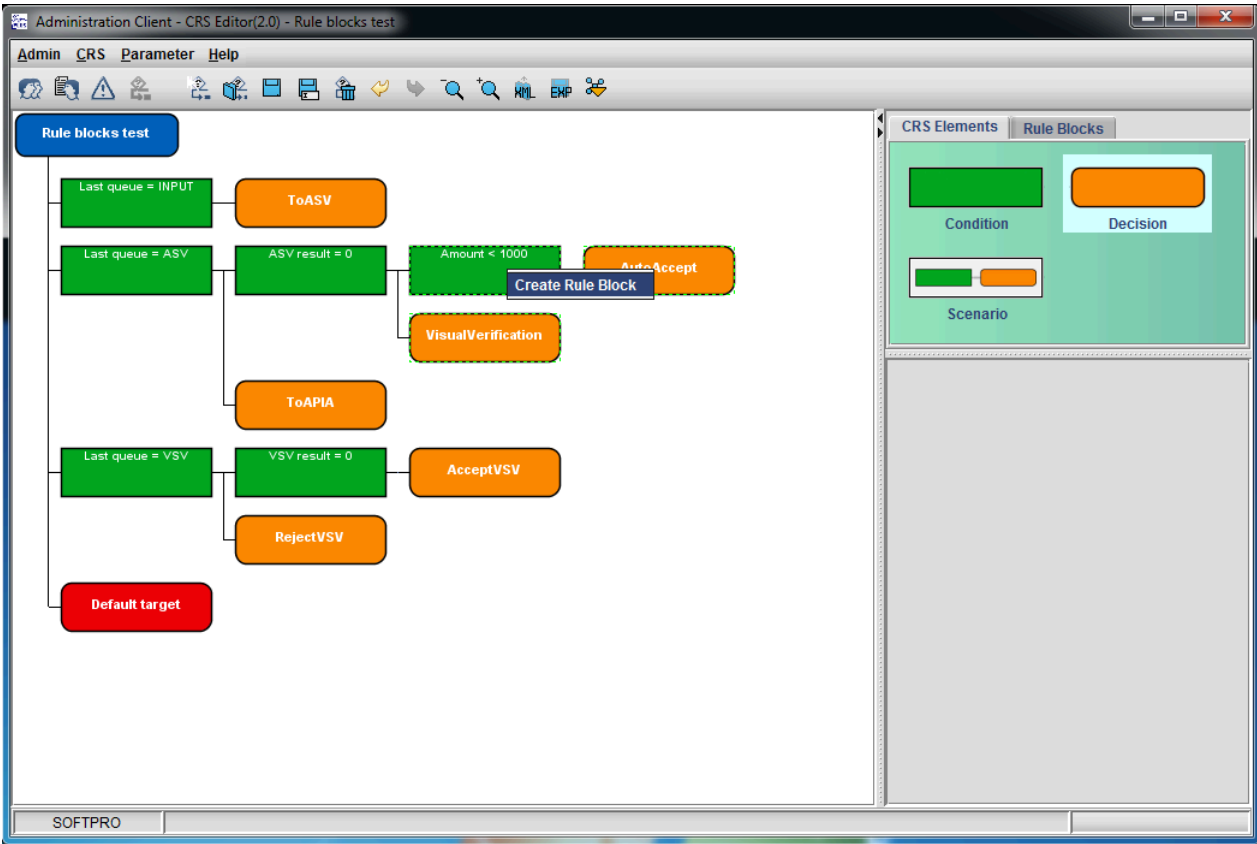


We also need to change the decision called **ToVSV** so that the items that did not pass ASV go to APIA queue for further verification:

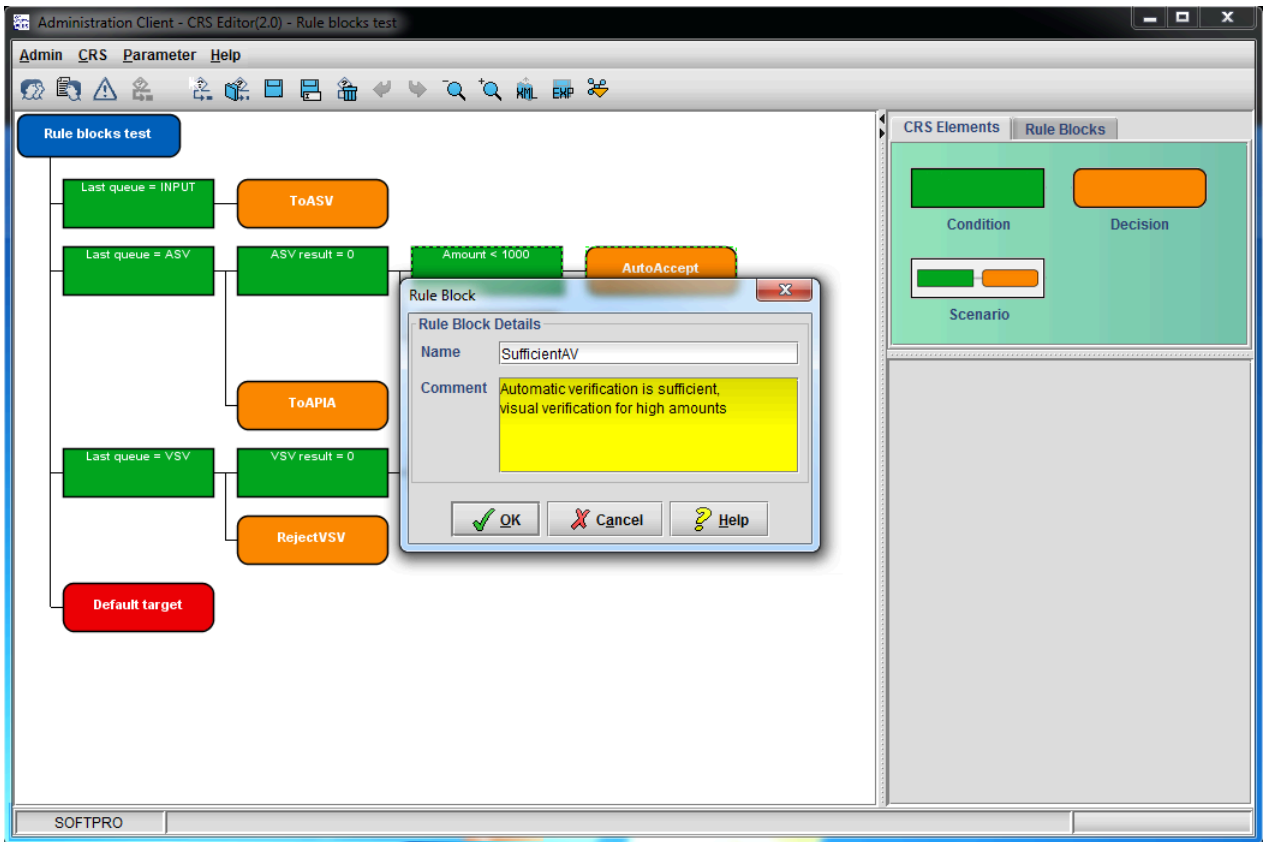


Now we are ready to create our first rule block. Let's select the **Amount < 1000** condition together with surrounding decisions with the mouse, then right-click on any element in the selection and choose **Create Rule Block** option in the context menu:





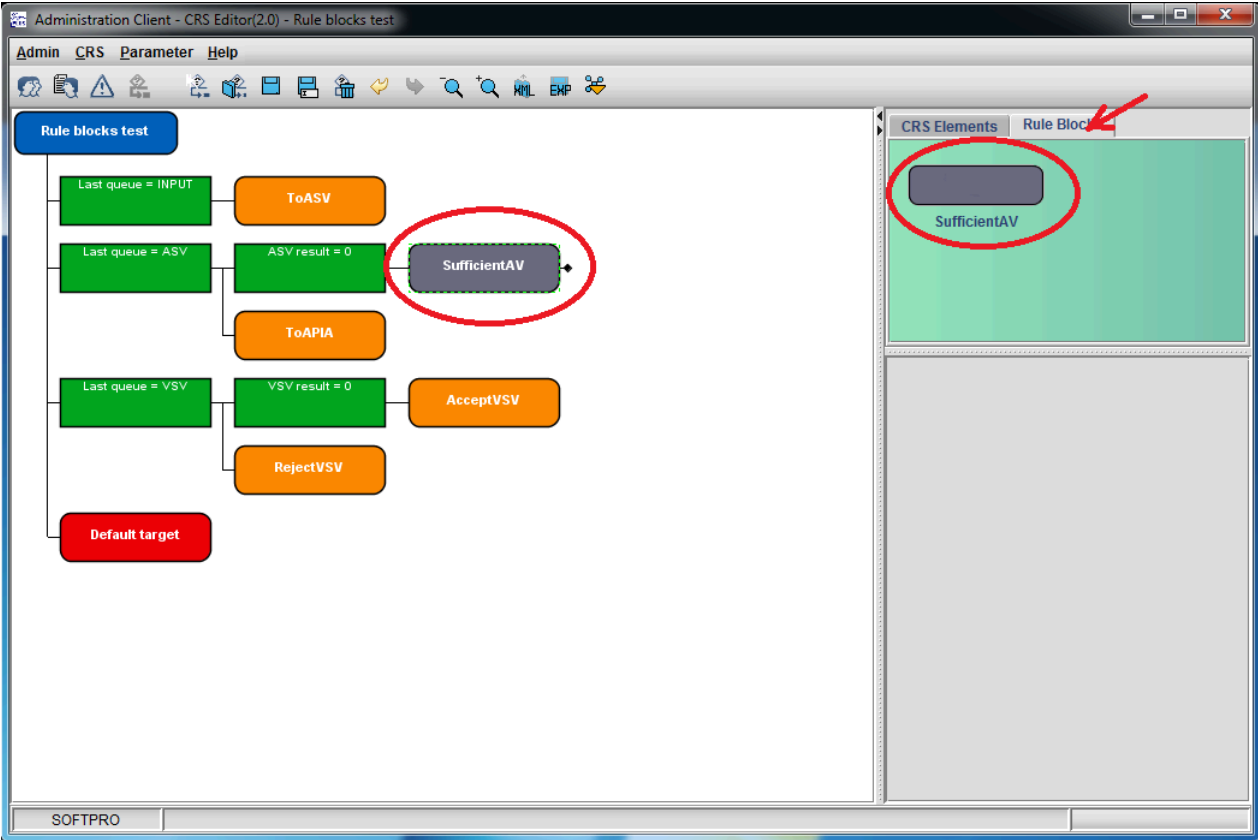
In the dialog enter an appropriate name and optional comment:



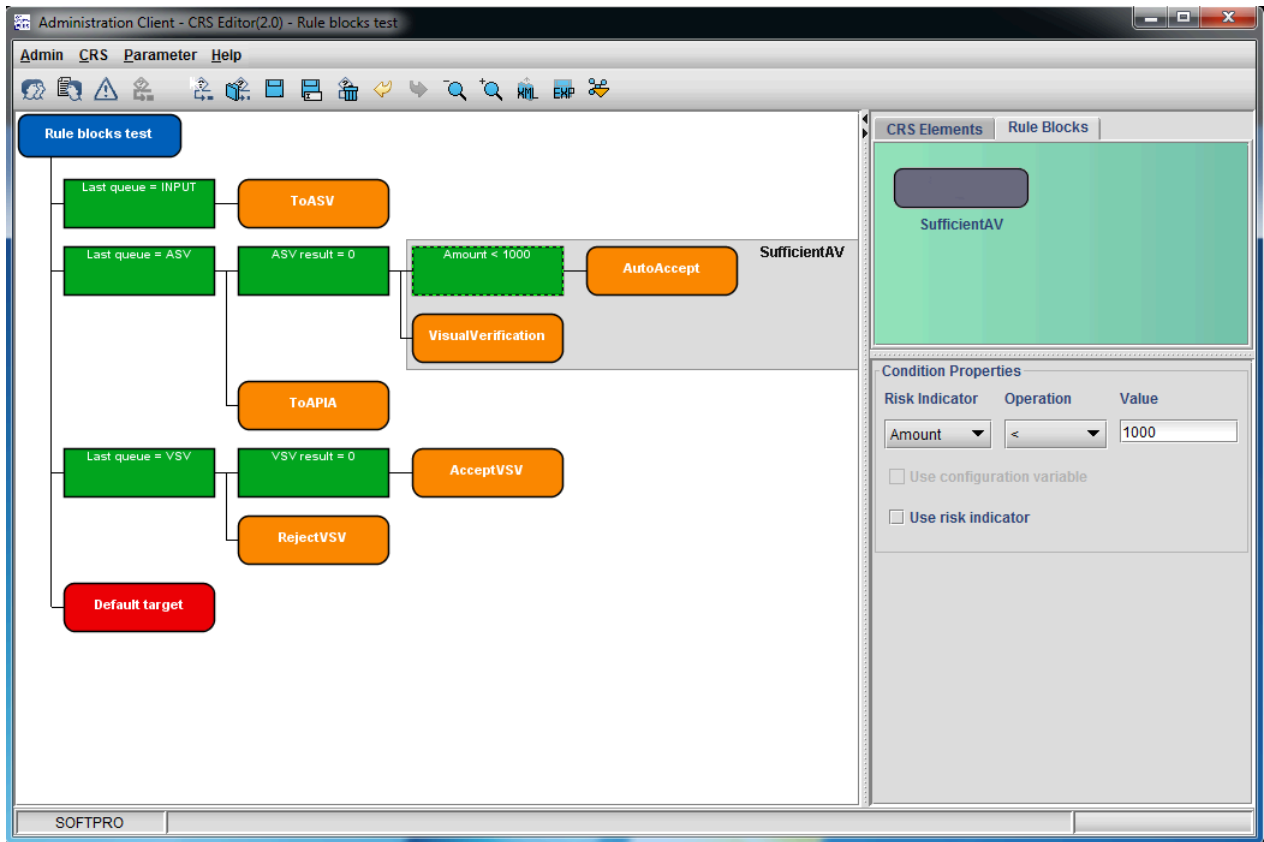
Our first rule block has now been created.

i Only continuous parts of the rule graph can be grouped to a rule block. It can be, for example, rule trees, or complete branches following each other.

On the **Rule Blocks** tab we can see the new element representing our new rule block that can be dragged and dropped to the rule graph in the same way as other CRS elements.

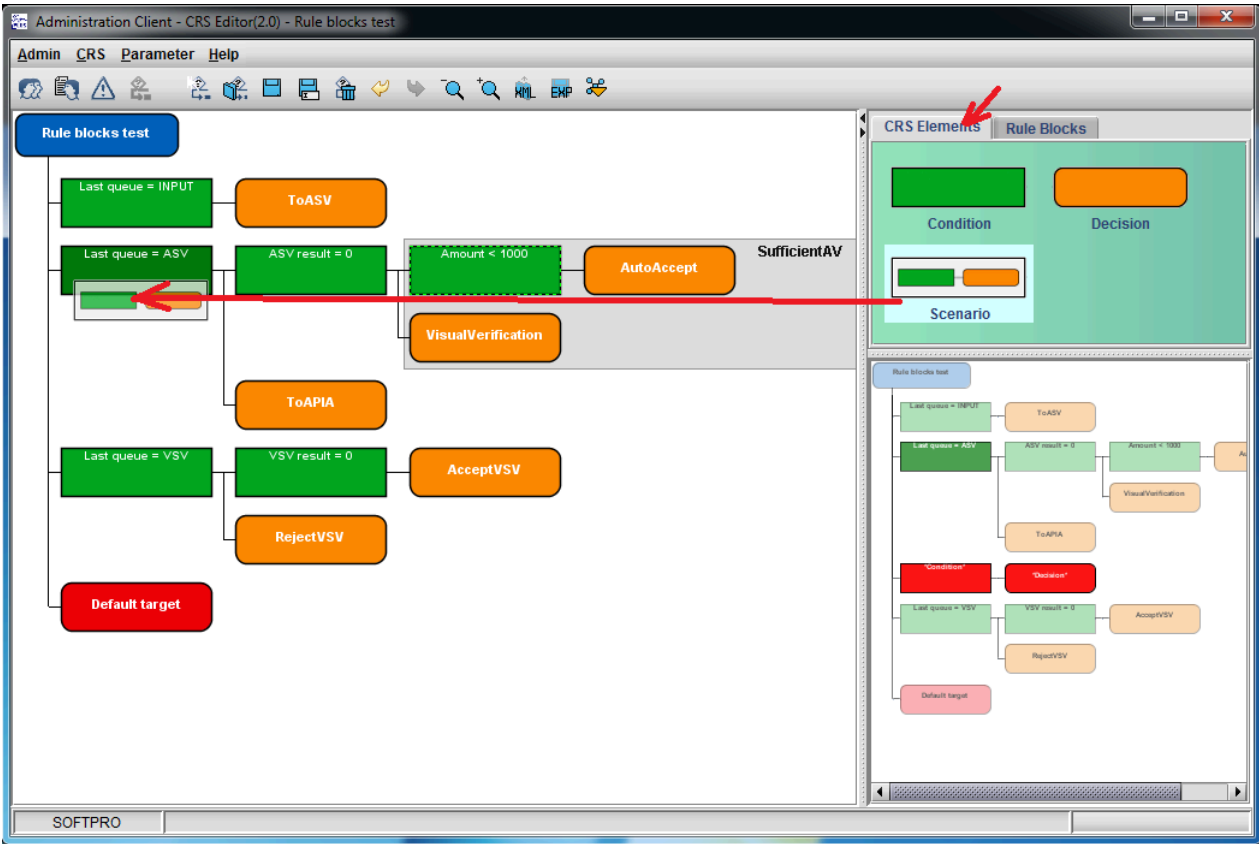


Let's expand the rule block by double-clicking the element in the rule graph and see the elements inside:

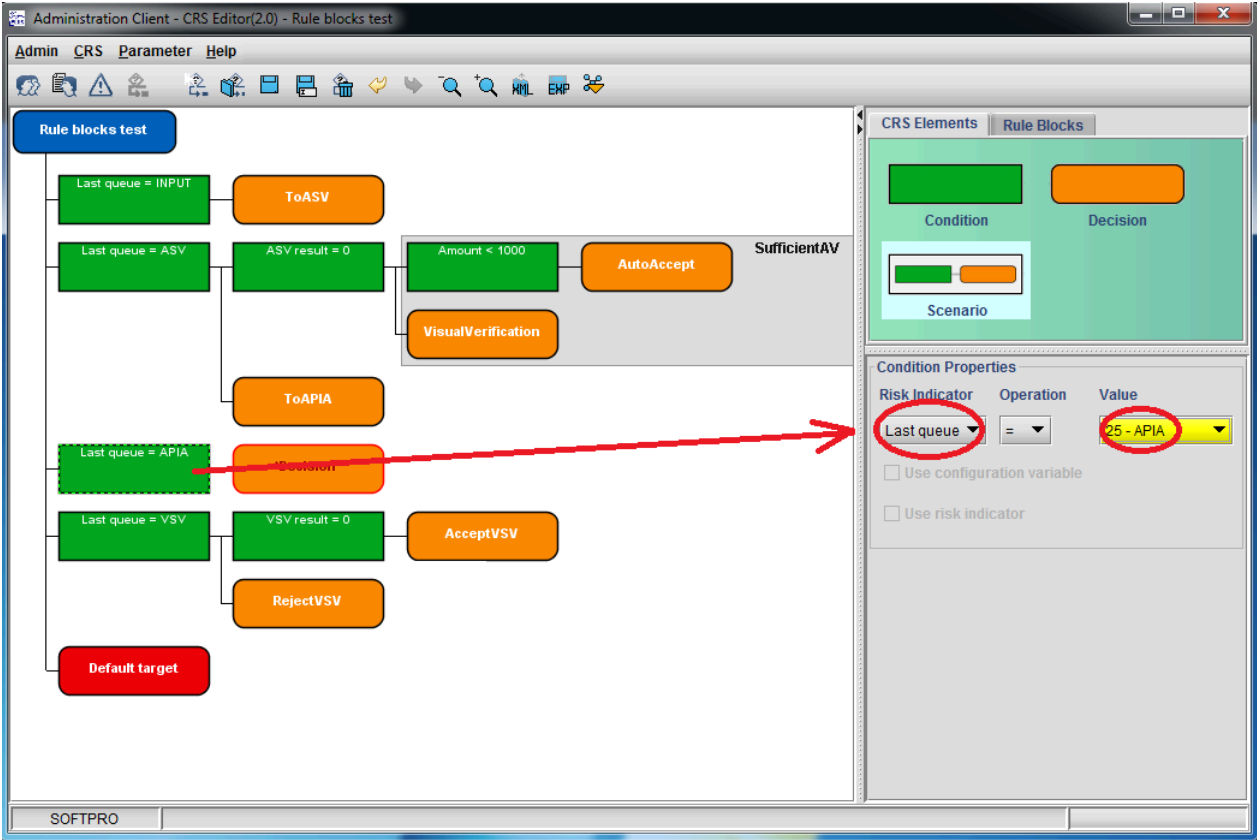


Add the rule block to another branch

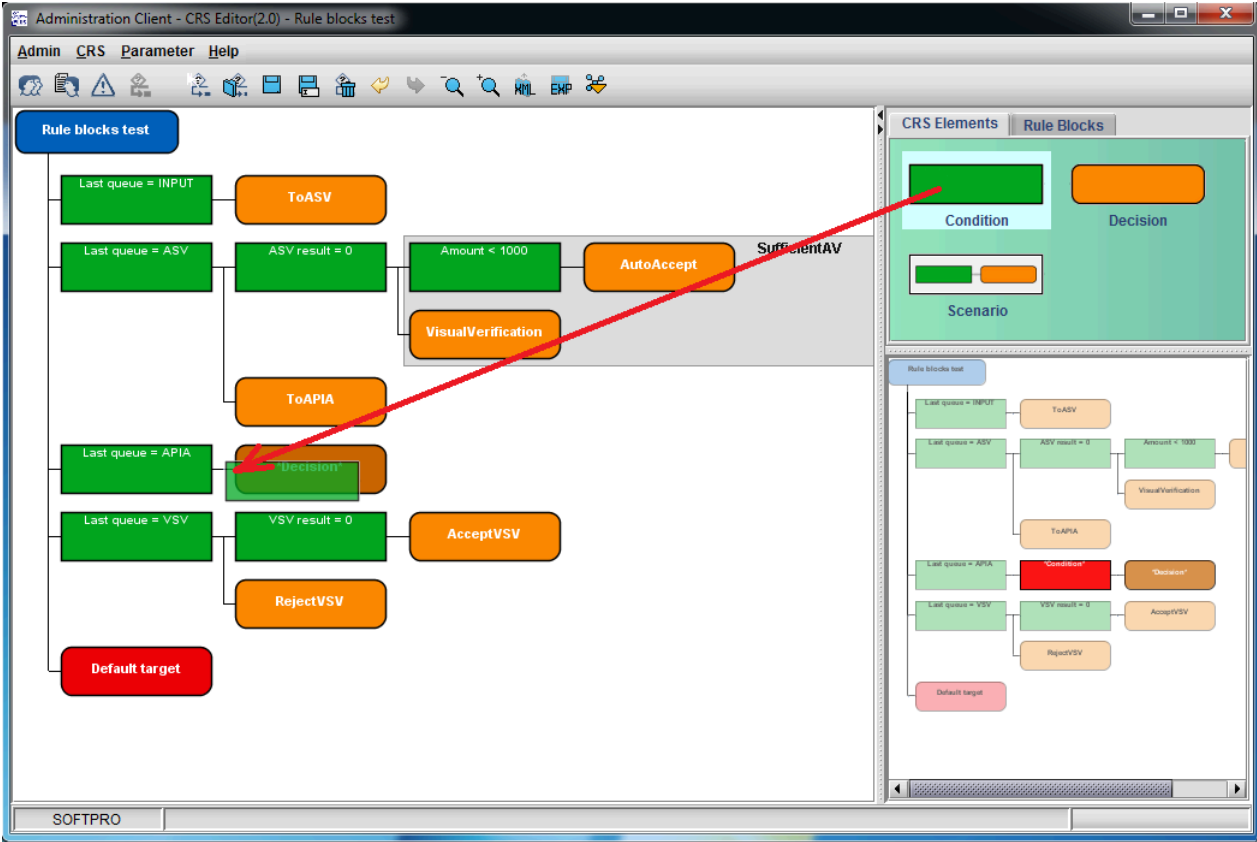
Now it's time to create a new branch for items coming out of APIA queue. As usual, we are going to drag and drop a scenario after the branch **Last queue = ASV**:



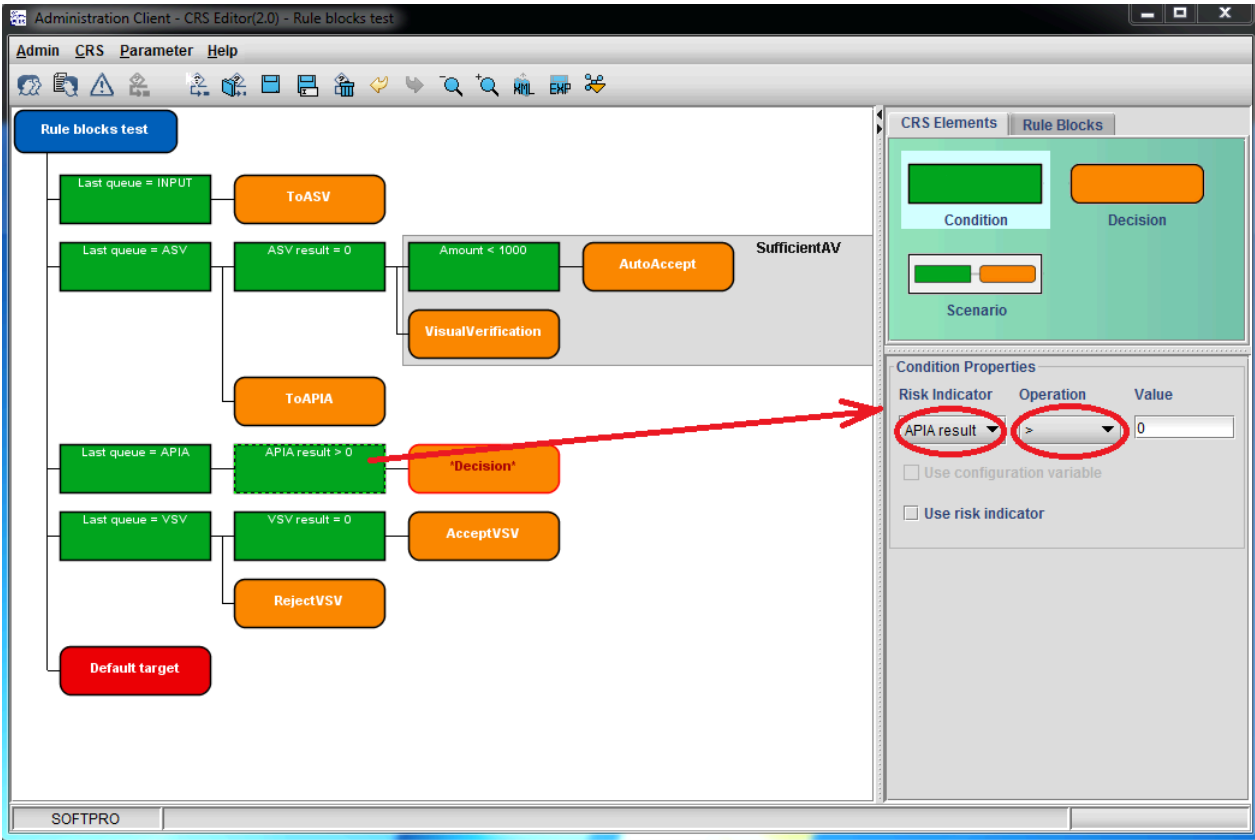
A new condition should filter out items coming from APIA:



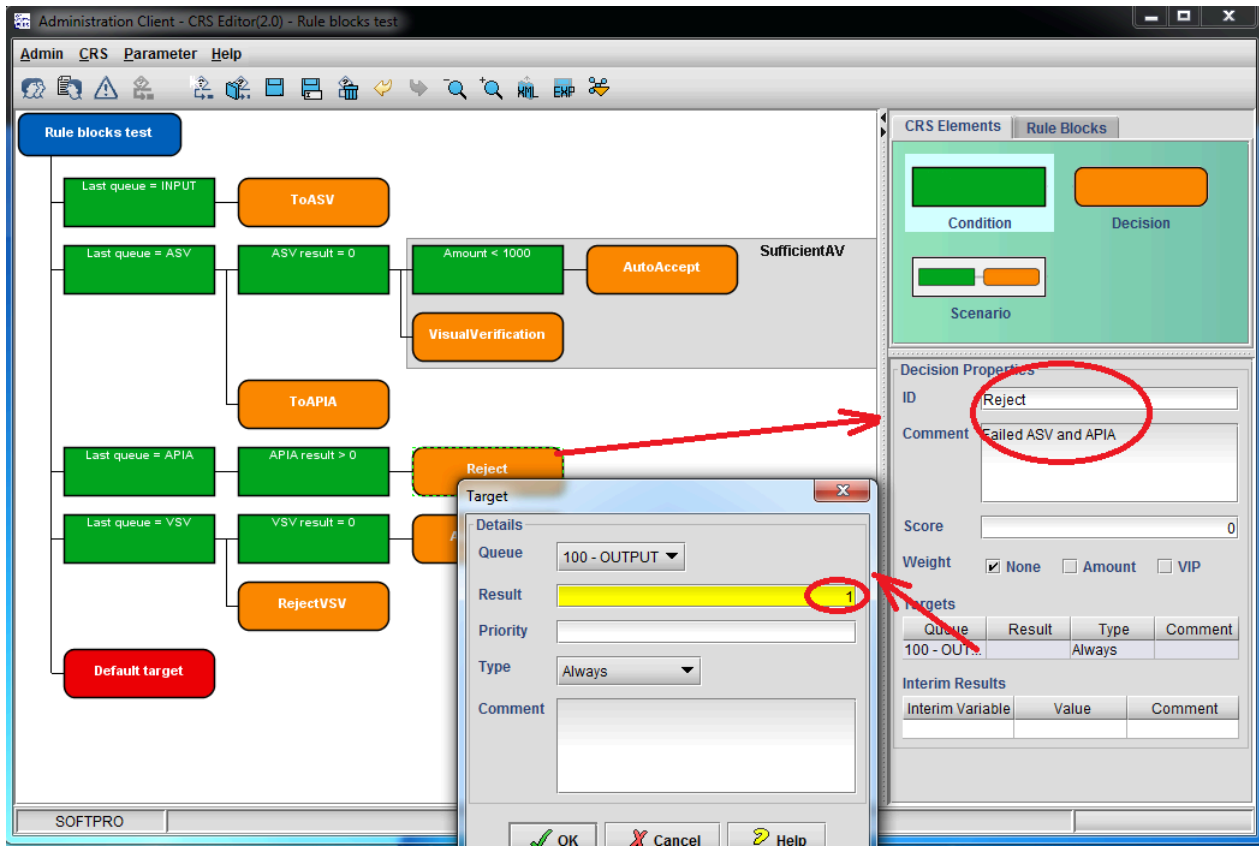
We need one more condition to check the result:



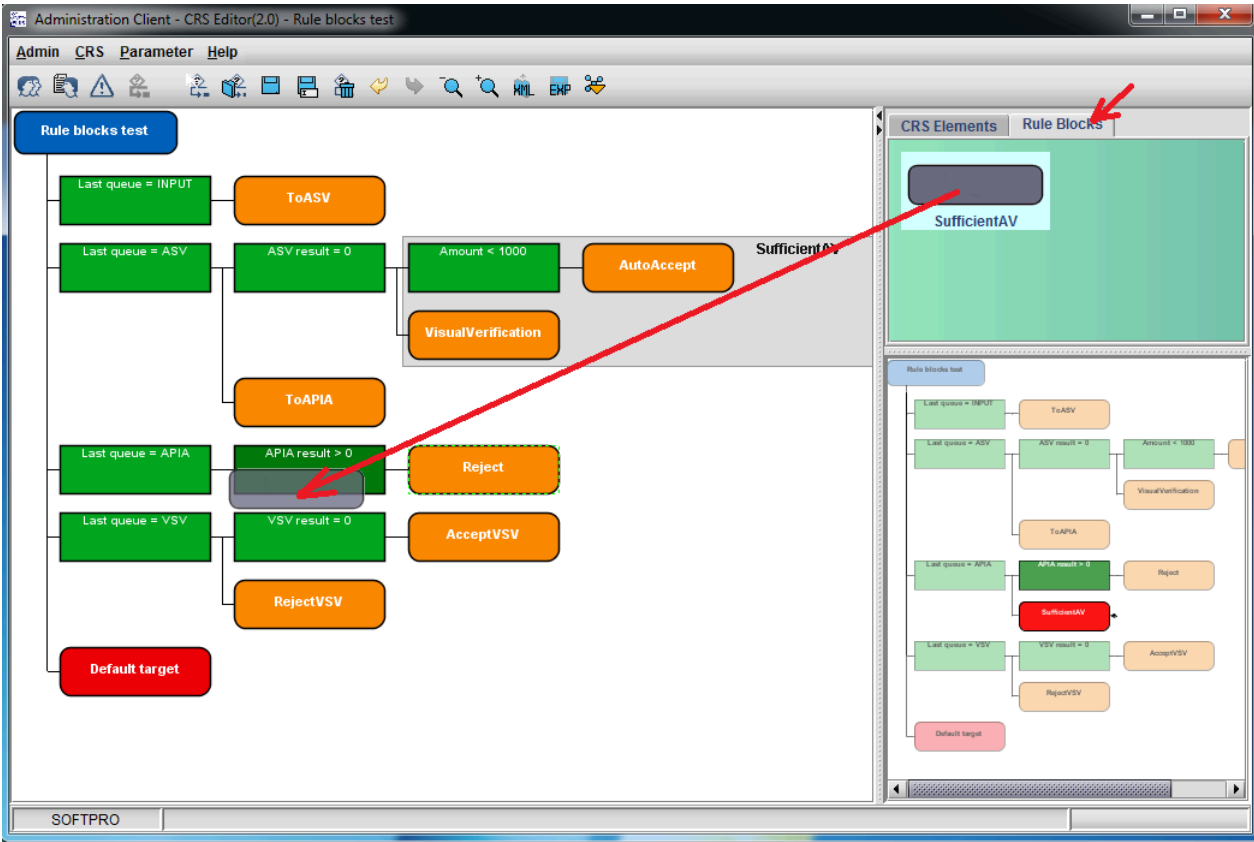
Let's first create the necessary workflow settings for the case when APIA returns a non-zero result:

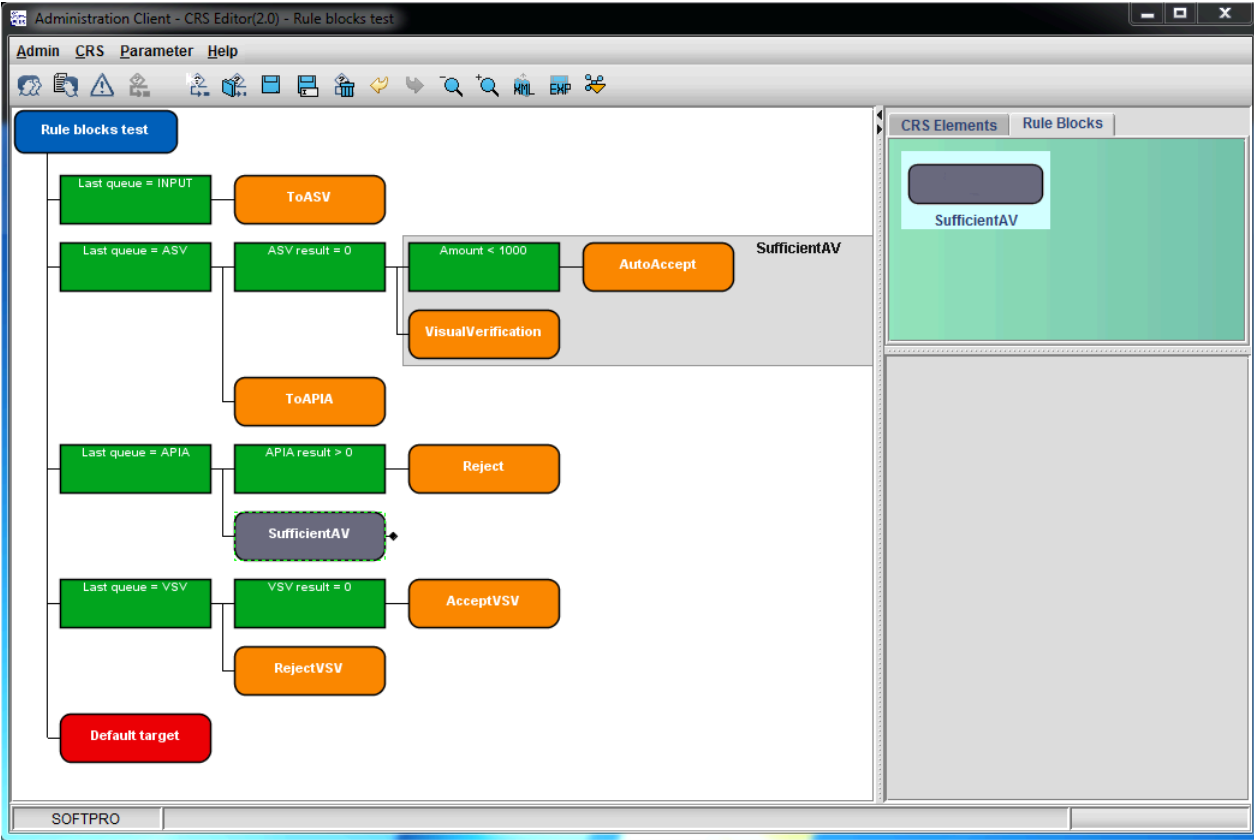


The last decision in this branch represents the case when an item failed both ASV and APIA. Let's reject this item:

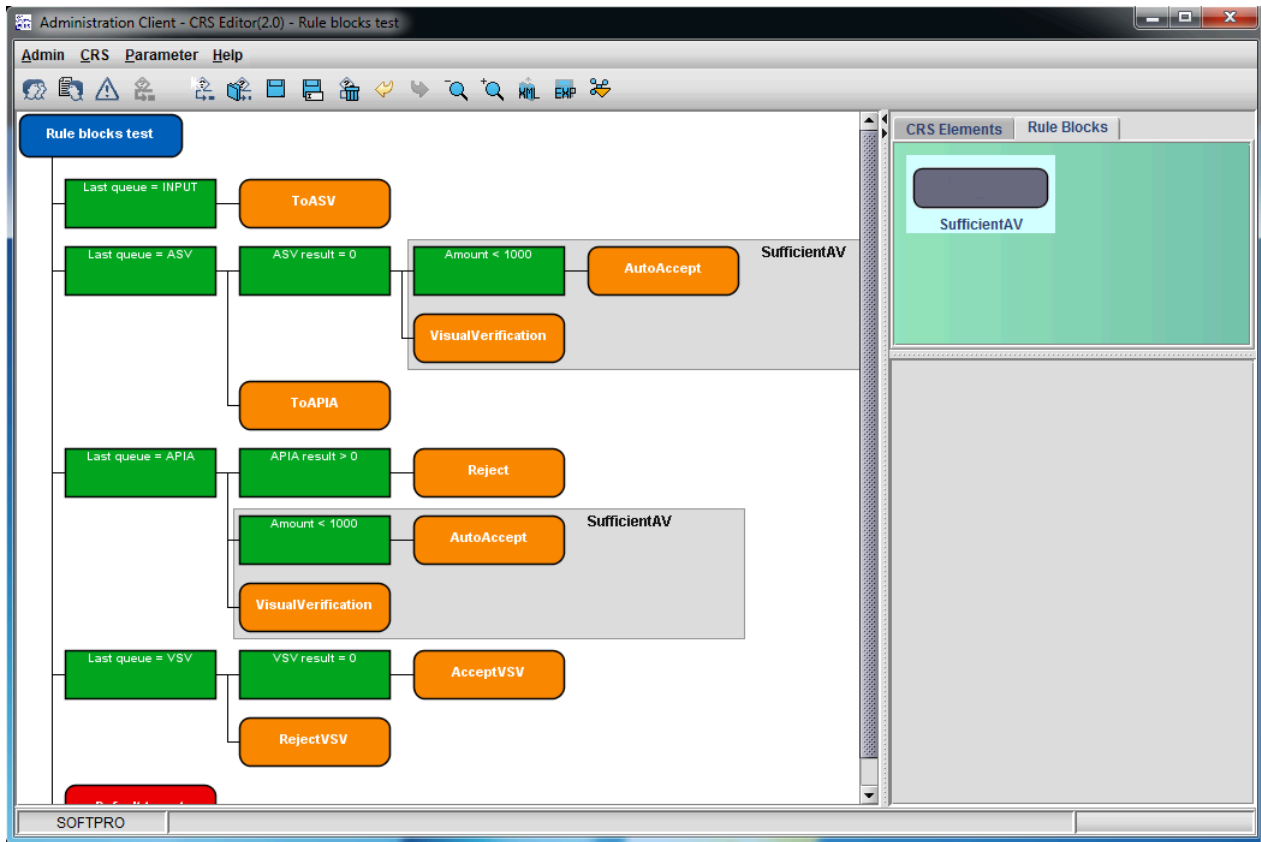


And what if APIA returns zero as result which means no error? Then we consider automatic verification complete and sufficient. The item's amount should be checked: items with small amounts are accepted, others should be verified visually. The logic needed here completely repeats the one we used in our rule block, so let's use it again:



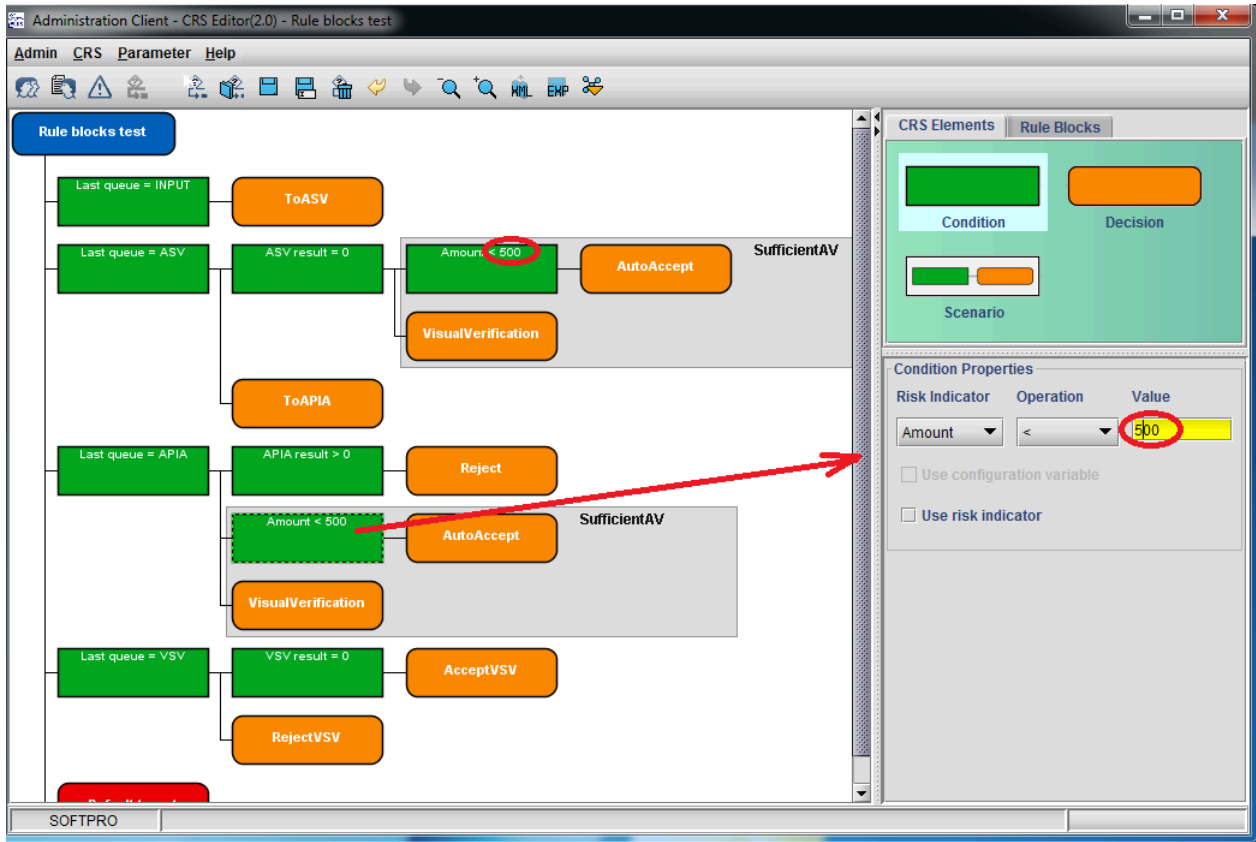


Now our workflow is complete. If we open the second rule block, we can see the same structure as in the rule block above:



Modify a rule block

What if we need to modify a rule block? For example, we need to update the boundary amount in condition element from 1000 to 500. With use of rule blocks we can just modify any instance as it was a regular element. All the changes will be immediately applied to all other instances of this rule block.



Changing the rule block structure is also possible, simply drag and drop an element into a rule block or use the context menu. Here is how adding a new condition looks like:

Administration Client - CRS Editor(2.0) - Rule blocks test

Admin CRS Parameter Help

CRS Elements Rule Blocks

Condition Decision Scenario

Decision Properties

ID: AutoAccept

Comment: ASV accepted items

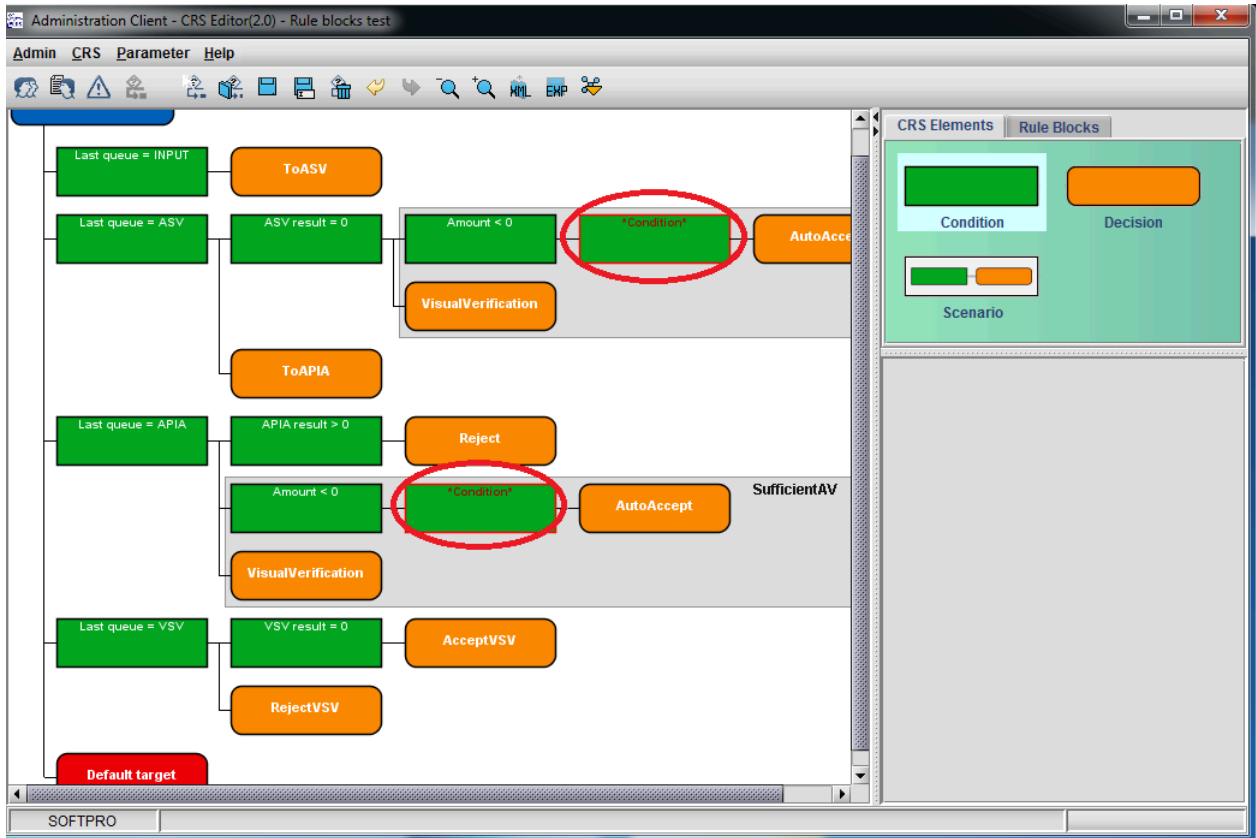
Score: 0

Weight: None Amount VIP

Queue	Result	Type	Comment
100 - OUT...		Always	

Interim Variable	Value	Comment
------------------	-------	---------

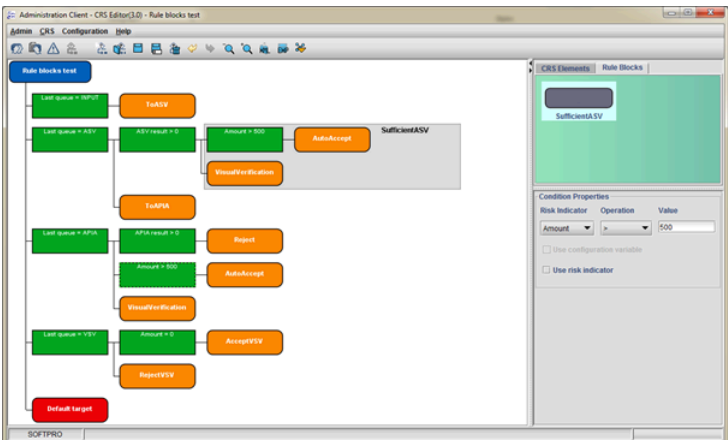
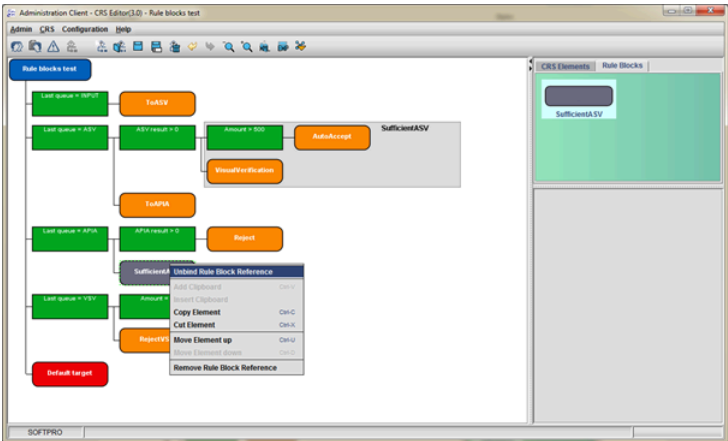
SOFTPRO



i Rule blocks can contain only basic elements and should not contain rule blocks.

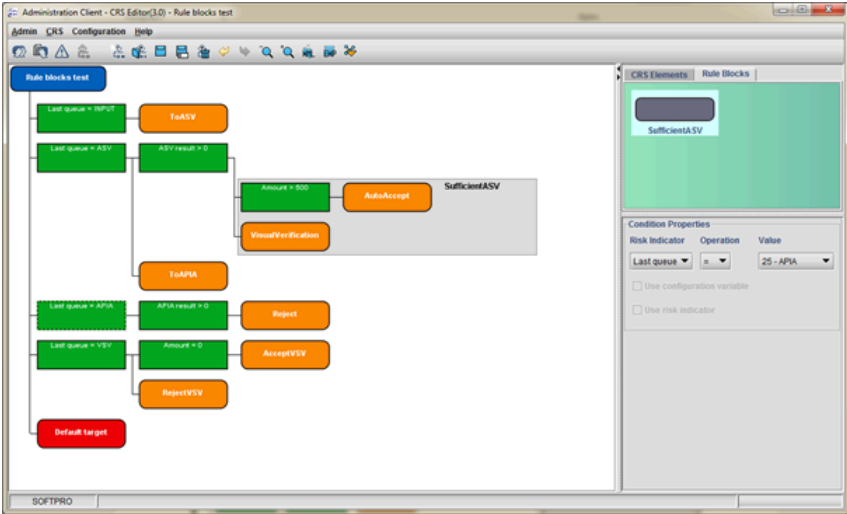
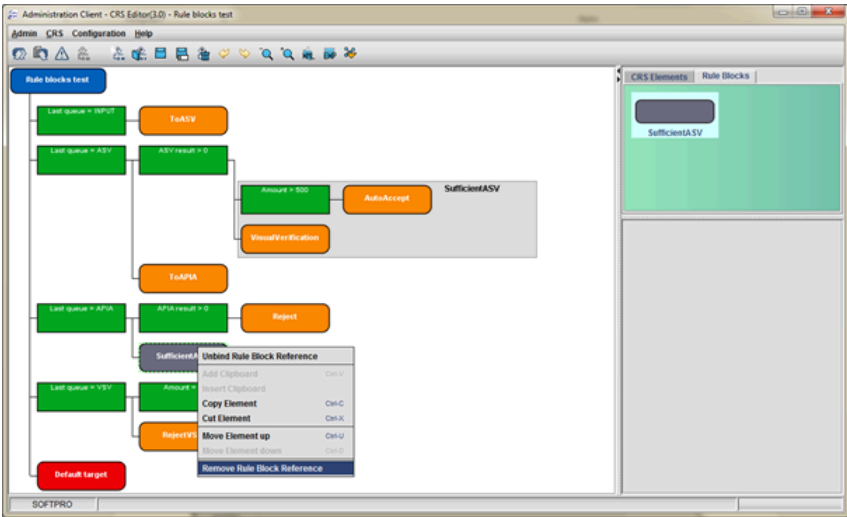
Unbind a rule block reference

We can just unbind a reference to rule block. Here we also collapse the rule block and select the particular entry from the context menu. We then keep all the single graph elements as own entities and not as a reference to a rule block anymore.



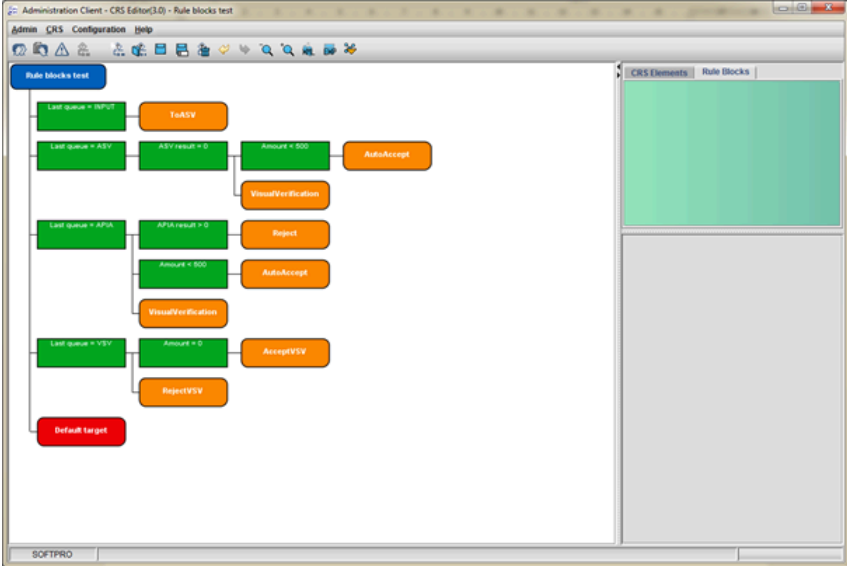
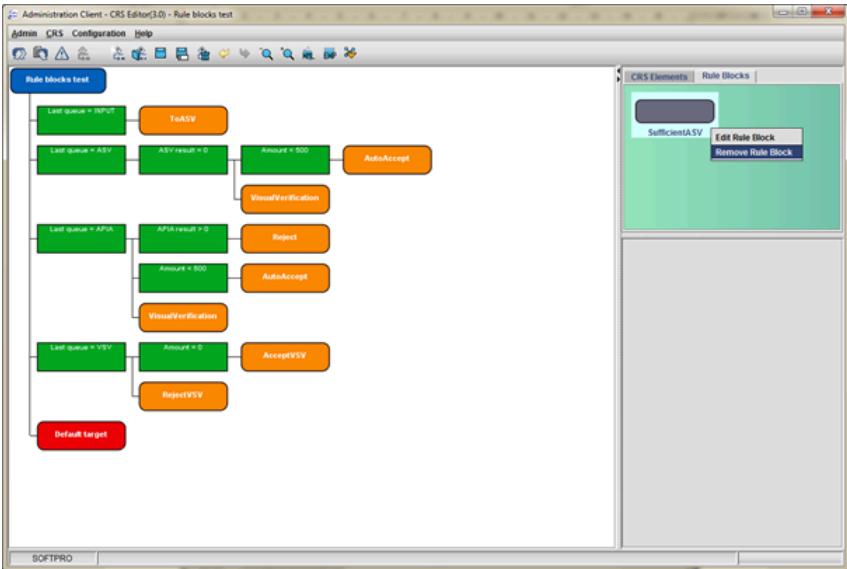
Remove a rule block reference

We can remove the complete reference to a rule block in simply collapsing it and selecting the particular entry from the context menu.

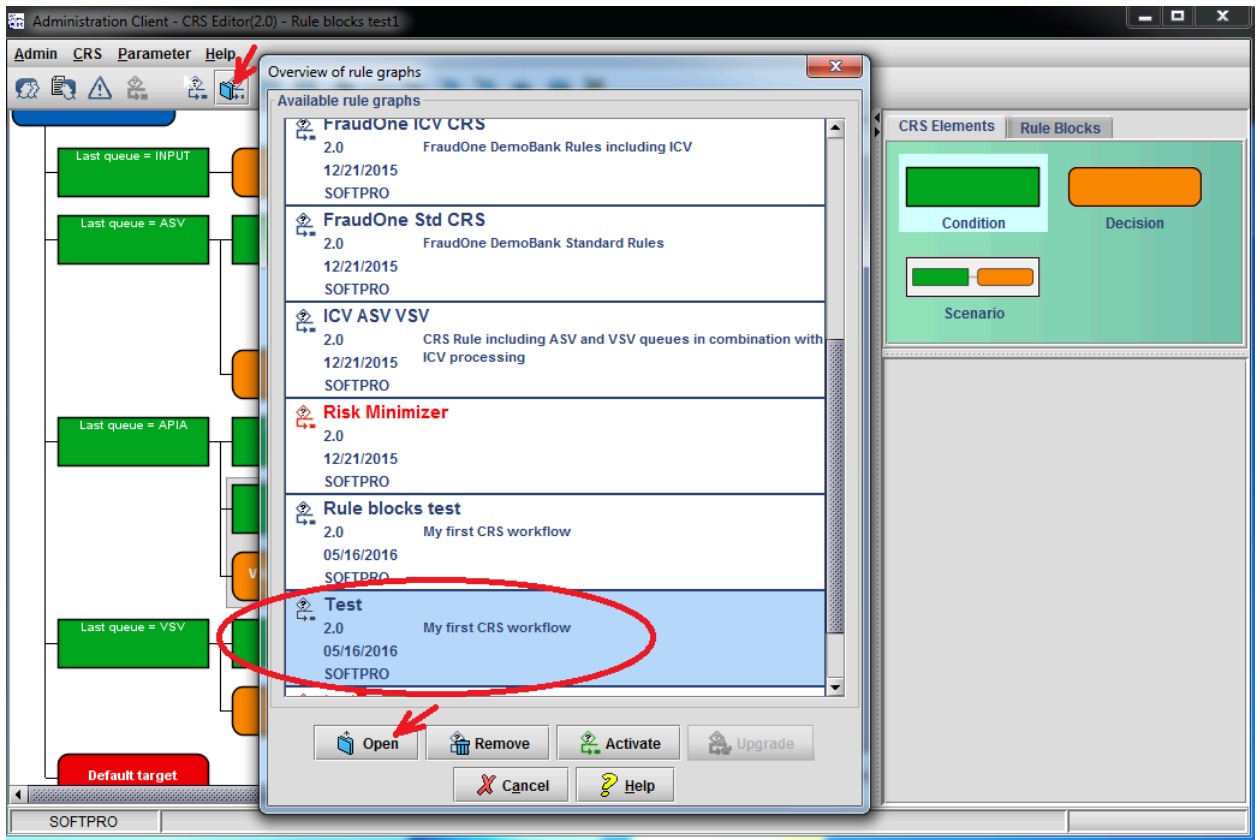


Delete a rule block reference

If a rule block definition is not referenced anymore, we can select the context menu entry **Remove Rule Block** in the **Rule Blocks** tab.



We can now save the rule graph and switch back to the one we stored at the beginning of the chapter:



Wrap up

Rule blocks are useful when a rule graph contains repeating patterns, so that such a pattern can be used as template. A new rule block can be created by selecting elements of the pattern using context menu. The selected elements should be continuous.

When a rule block is defined, it appears in the **Rule Blocks** tab and can be dragged from there to the rule graph as usual CRS elements. Modifications of such a rule block reference, both structural and configurational, are immediately applied to all other rule blocks of the same type within the whole graph.

A rule block cannot contain itself or other rule blocks. A rule block reference can be unbound, what means that the elements building the rule block are not representing references to a rule block template anymore. A rule block reference can be removed from the graph. If no rule block references are contained within the current graph, the rule block template can be deleted from the **Rule Blocks** tab.

Chapter 5

Score, risk and priorities

Basics

Up to now you only used CRS to move items around. But what about that "R" and "S" in the name? "Risk" and "scoring" have not been covered so far.

CRS does not only handle items but it is also meant to assess risks and classify items. When visually reviewing items, you likely want the most important items shown first. The question here is: What is an important item? Even though there are a couple of common principles, most customers will answer that question differently. This is why the main focus has been put on flexibility and versatility. You'll find that you can achieve the same goal in different ways. The following paragraphs describe the design concepts and some simple solutions to sample problems.

When you try to sort your checks there are two things you are looking at. You are creating CRS rules to determine how probable it is that a check is fraudulent. On the other hand, a \$50 check that is 80% fraud will probably hurt less than a \$100,000 check that you are only 20% sure it is fraudulent. Therefore, the CRS is designed to combine information when items are rated:

- **Score**
The score is determined by the CRS rule that catches the item. It tells the probability that this check is fraudulent.
- **Weight**
The weight is determined by the importance of the check and is not dependent on the fraud probability. Factors used to calculate the weight, are the amount and the fact that the check belongs to an important customer, or not (VIP indicator).
- **Risk**
The risk is the total "how much does this check hurt" indicator. It combines the score and weight defined above. The higher the fraud probability (score) and the higher the item weight, the riskier the item will be. This is what is used to prioritize and decide items.

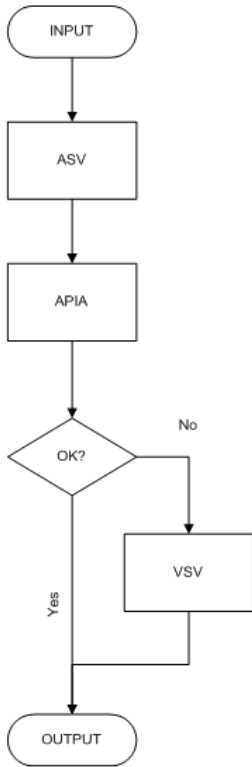
Some preparation

We will change our CRS rules slightly to give us a new engine to play with.

Start with the rule set used in the previous chapter. Open the CRS editor and load the **Test** rule set.

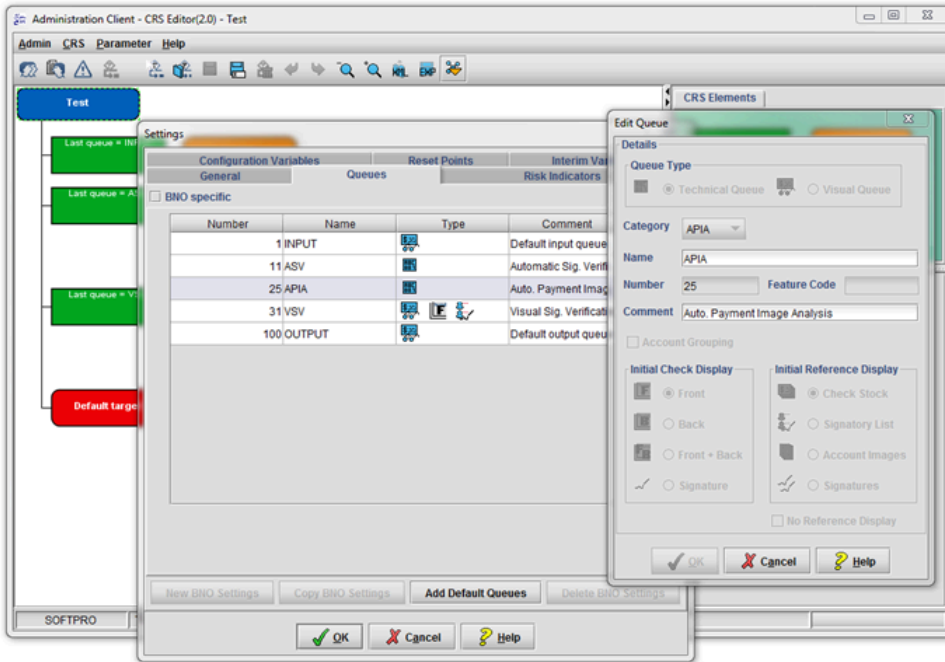
To have an additional result to work with we will add check stock verification to our workflow. Automatic check stock verification is done by the APIA engine. It works similar to ASV. You have a

set of servers and another one of automatic verification engines using them. In terms of CRS you only need to provide a queue they can work on. We will add the APIA queue after the ASV queue.

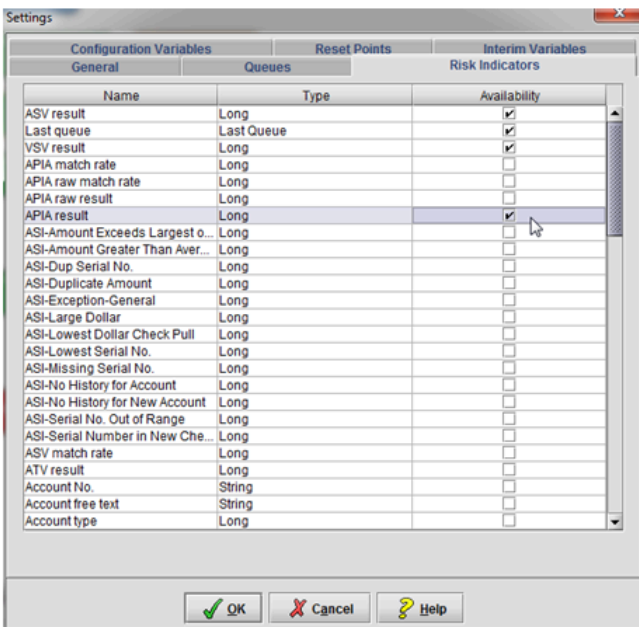


Items coming from signature verification will be routed directly to the check stock verification.

Open the properties of the current rule graph and go to the **Queues** tab and add a new APIA queue. You can do so by clicking the **Add Default Queues** button or configuring the queue manually.

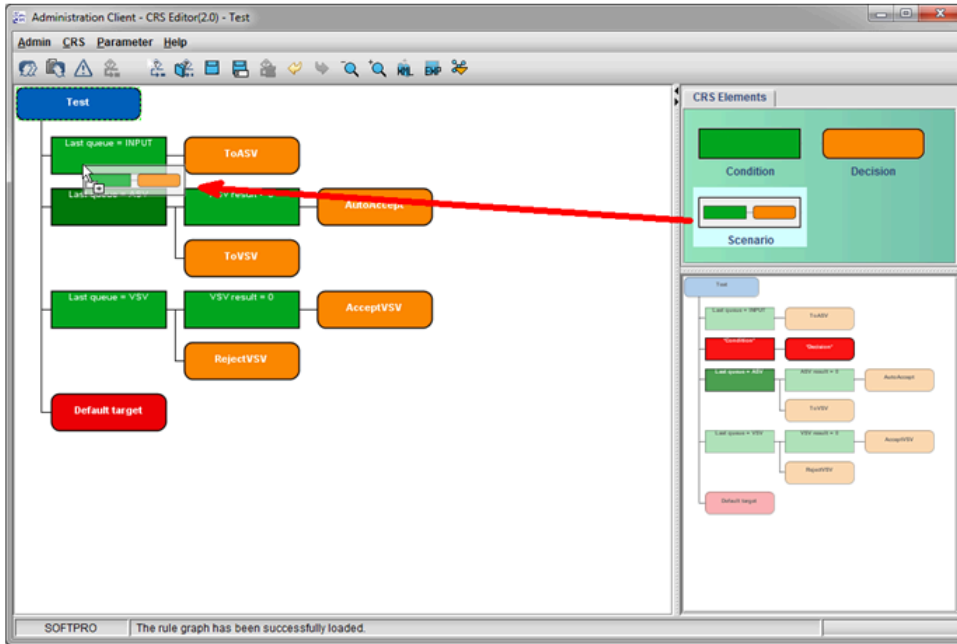


Since we want to use the APIA result for our scoring, you will have to add it to the list of risk indicators used by CRS:

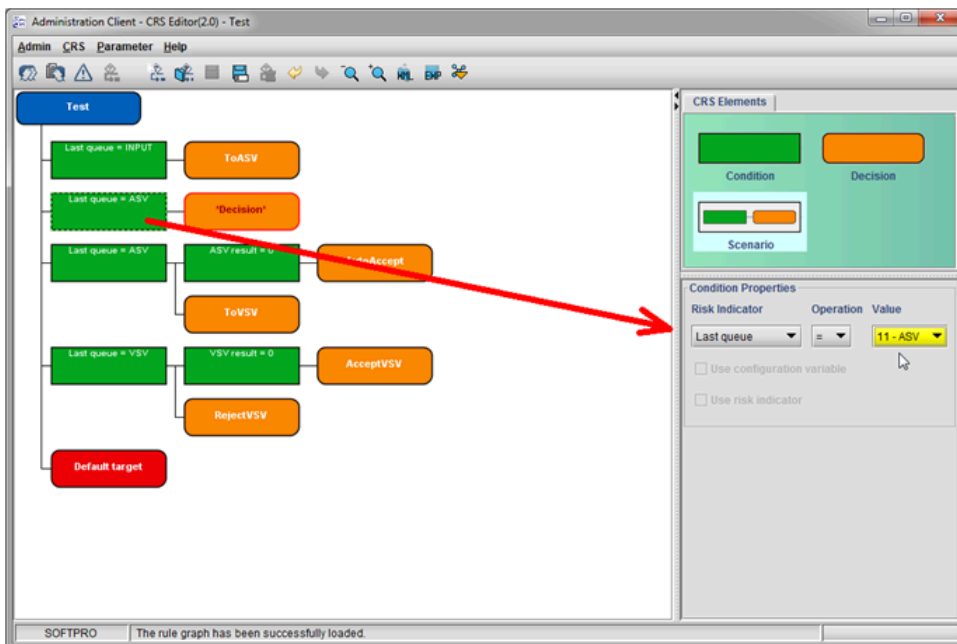


You can now return to the rule graph and include the APIA queue into the routing. First, you will add a new scenario for the ASV queue. It is easier to do this than modifying the existing one, since you

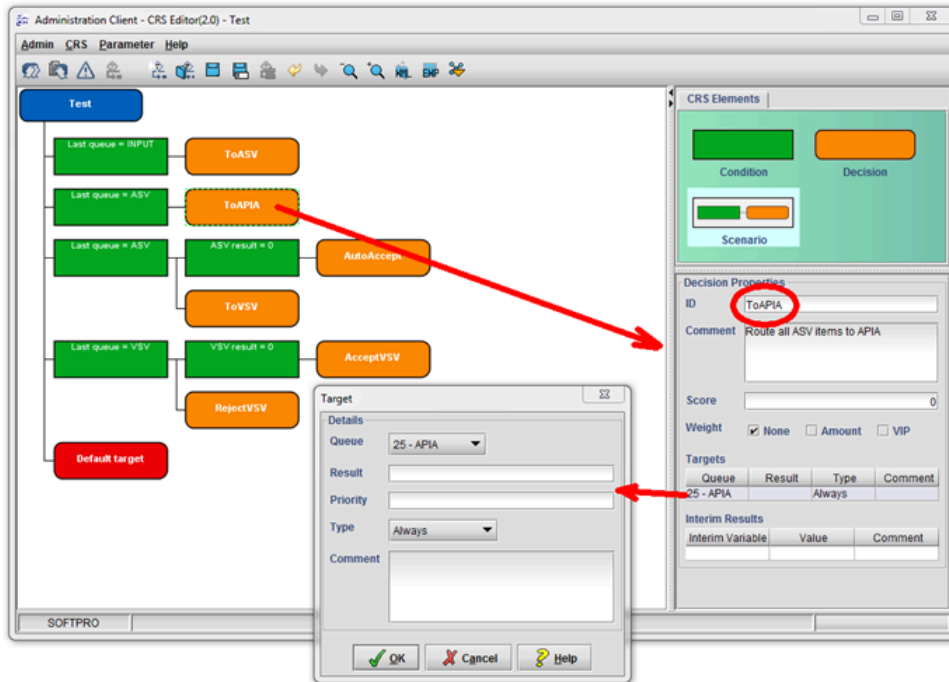
can reuse that for the APIA queue. Insert a new scenario immediately after the one starting with **Last queue = INPUT**:



Change the condition to read **Last queue = ASV**:

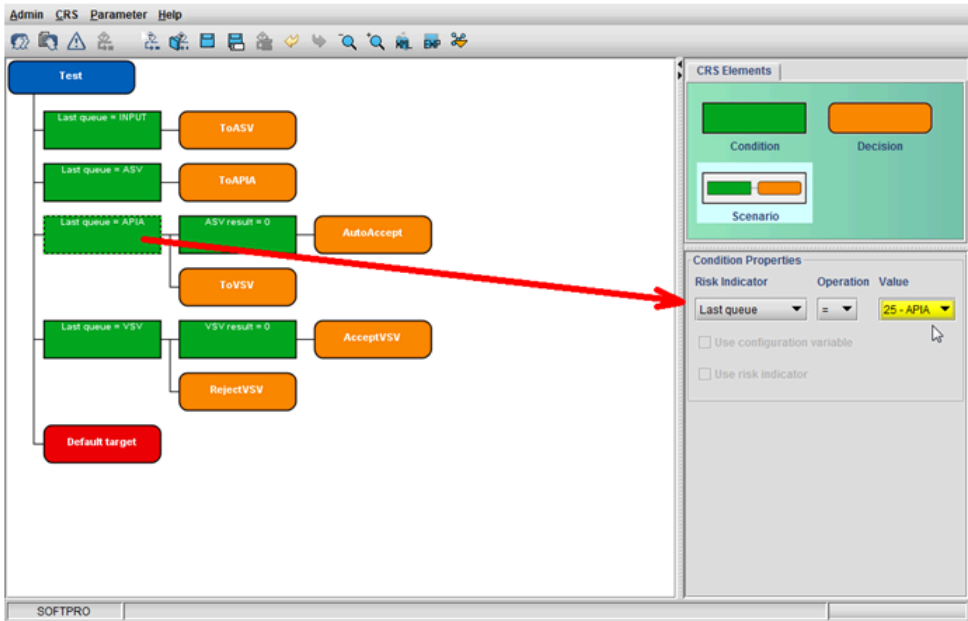


Edit the decision and route all items coming from the ASV queue (that is what **Last queue = ASV** means) to the APIA queue. Click on the decision, set the **ID** to **ToAPIA** and change the target to the APIA queue:

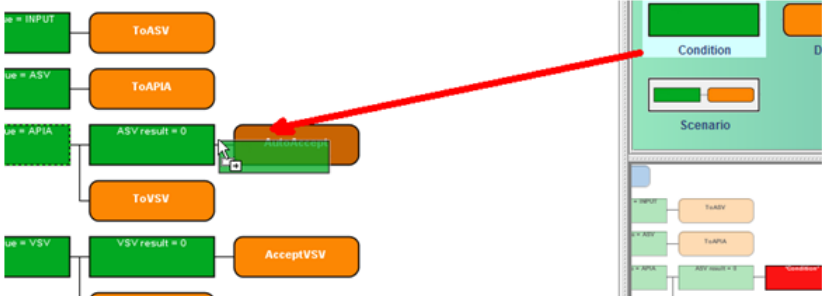


The first half is done, items from ASV will be routed into our new queue. Now we will have to route the items that have completed APIA. We will do this the same way as we routed the items from ASV previously, only we now have two results to consider.

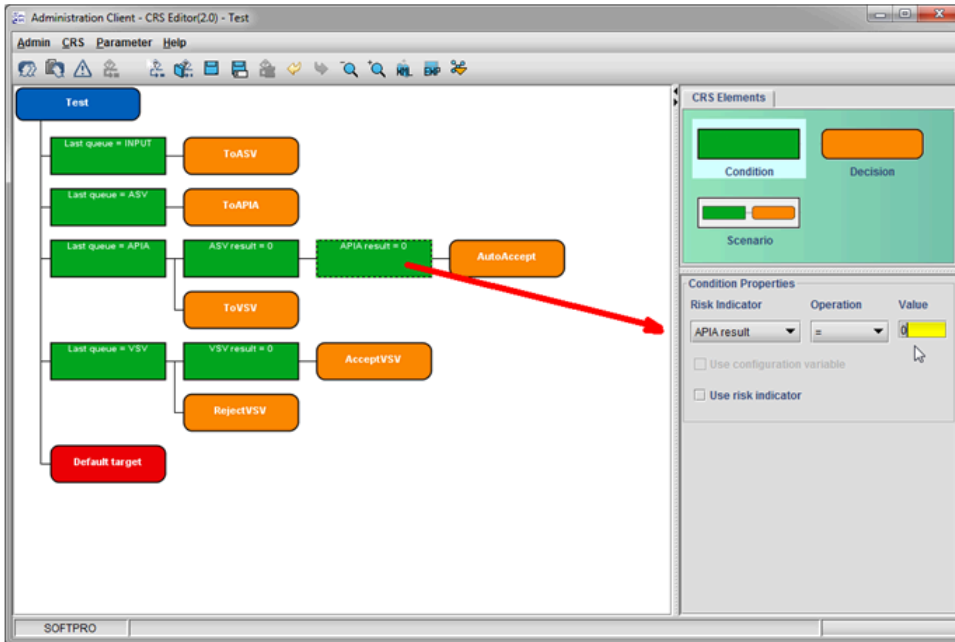
If both our automatic engines consider the item to be ok, we will auto accept it and route it to the output queue. If any of the engines considers the item faulty, we will send it to visual verification in the VSV queue. Looks similar? This is why we kept our old ASV scenario. You can now reuse it. First change the **Last queue = ASV** to read **Last queue = APIA**. Make sure you get the right one. You want the old one to be changed (the second one if you followed the instructions):



You also want to consider the APIA result in addition to the ASV result. Add a new condition between the **ASV result = 0** condition and the **AutoAccept** decision:



Change the newly added condition to read **APIA result = 0**:



Now the **AutoAccept** decision at the end can only be reached if both conditions **ASV result = 0** and **APIA result = 0** are true. If any of them fails (if any automatic engine rejected the item) the check will be routed into the VSV queue instead (fallback to the **ToVSV** decision).

You now have everything needed to get started with scoring. Saving the CRS rules at this point would be a good idea.

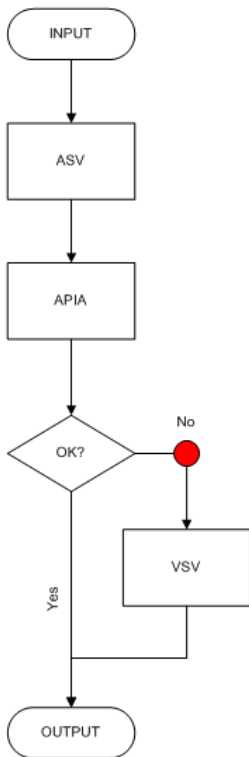
Score

This section will first highlight the score. The goal is to sort items by their fraud probability.

The first question is where (when) you want to do that. There is no reason to sort items that are going to be processed by ASV or APIA, since they are going to all be processed automatically anyway.

i That might not be correct for everyone. If you don't have time to process all items during the day, you might want to sort automatic items by, say, amount, so you don't miss important items.

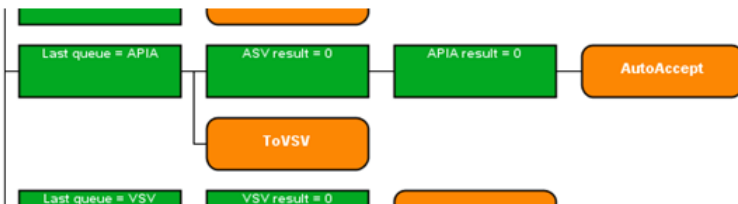
However, you may want to sort the items that are presented for visual review. Most users usually prefer to see important checks first.



The rules that need to determine the score are the ones that move the items to VSV after ASV and APIA completed, since it is VSV where you need to have the items sorted.

i Customers that use a non-visual workflow (engines only, review in external systems) will want to rate and score the items in the OUTPUT queue and use that as a starting point for their own visual review. The concepts work the same way.

Review the current rules that move items to VSV:



The scenario that applies to items coming from APIA will either auto accept the item if both engine results are good or send them to VSV (**ToVSV** decision) instead.

Obviously, it is this decision that needs to do the rating. But since you only have one rule that moves all items to VSV there will be no big difference in scoring. That will need to be fine-tuned a bit.

First question: What do you use to score items? Answering this question is very complex for a production system. You can use any item information like engine results and match rates, account

information like age of the account, demographic data such as ZIP codes, you name it. This is why we carry all those risk indicators available per default. For the simple example here you will do only a very quick scoring by rating items that failed both ASV and check stock the worst, followed by items that failed only ASV, followed by items that only failed APIA and rate items with two good results best.

Since items that have two good results do not even make it to VSV you can leave those out. For the other three cases you will need three new rules.

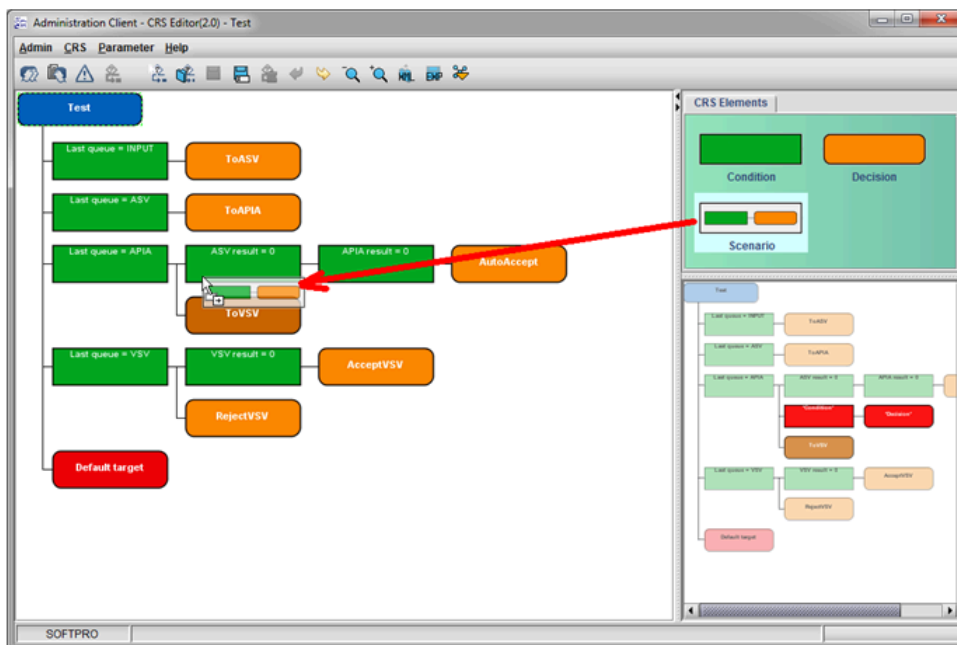
First select the **ToVSV** decision and delete it by right-clicking and selecting **Remove decision**.

Now you start inserting the new rules. But which one do you need to do first? Does rule ordering matter? Remember the discussion on how the parser works. It will try rules from top to bottom, left to right and stop at the first decision it reaches. If you put the **ASV bad** rule first and the **ASV bad and APIA bad** second, the parser will stop at the first one even for items that have both bad ASV and APIA. It will never reach the second rule.

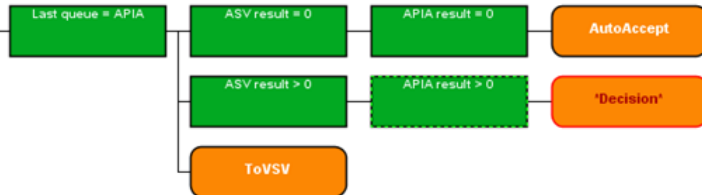


Ordering is important. You will need to put the most restrictive rules, the ones that give the highest scores, first.

The first rule will thus be the one that catches items with two bad engine results. Add a new scenario below the **ASV result = 0** and the **ToVSV** decision:

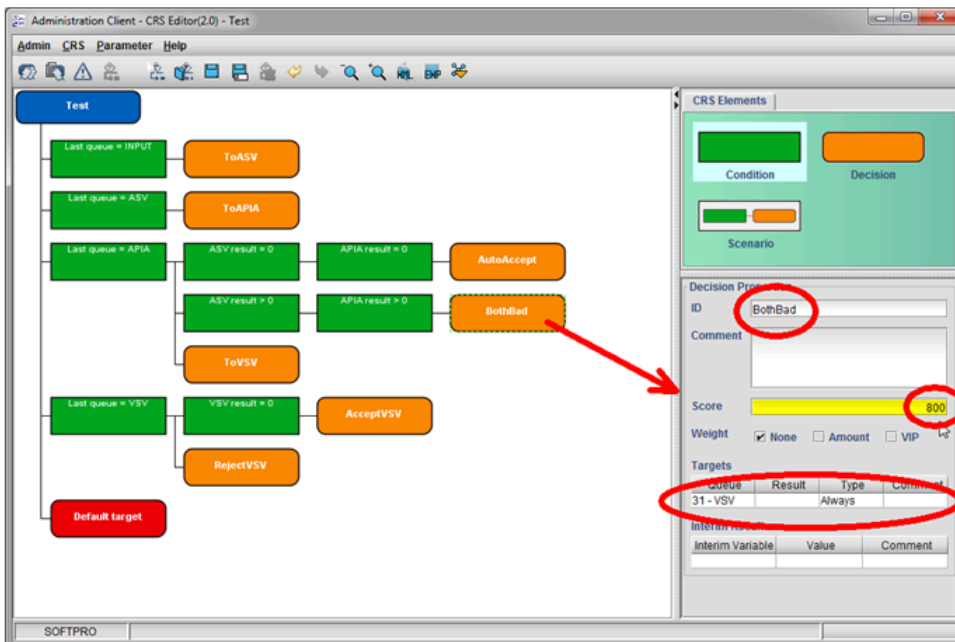


A bad engine result is any that is not 0. You will probably want to handle some result ranges differently in production (a missing account or reference is quite a bit different from a signature mismatch). Change the first condition to read **ASV result > 0**. You will need to set the risk indicator to **ASV result**, the operation to > and the value to **0**. Then add another condition and change it to read **APIA result > 0**. Now this rule will be true if both engine results are bad.



What do you need to do with those items? You will want to move them to VSV, but you will also want to give them a high score. What a high score is, is something for you to determine. For this example 800 for is used for both results being bad, 600 for bad signature and 200 for bad check stock.

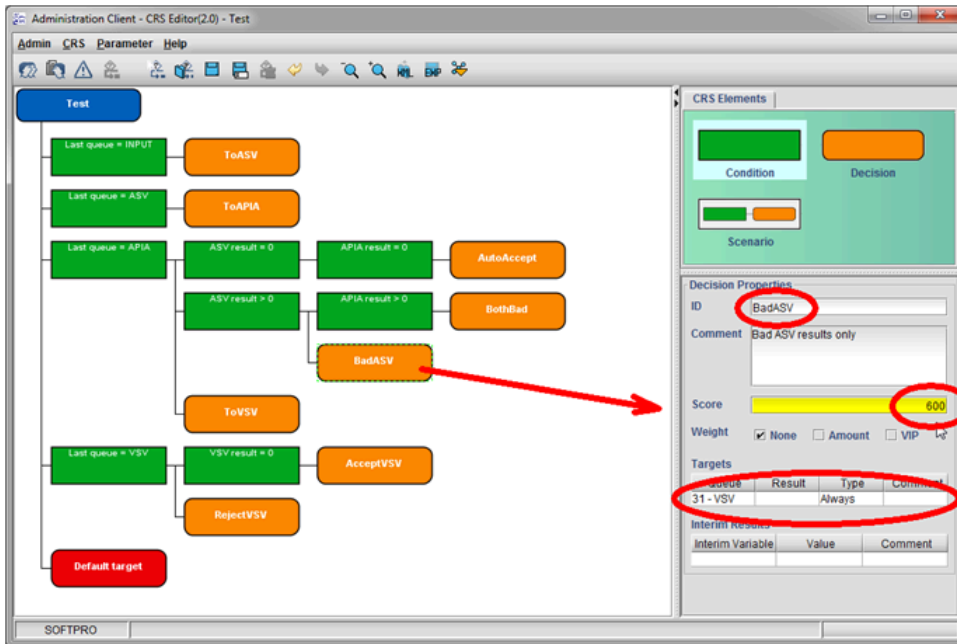
Select the decision at the end of the rule. Change the name to **BothBad**. Change the target queue to **VSV**. Set the score to **800**. Leave the weight to **None**. That will be discussed in the next chapter.



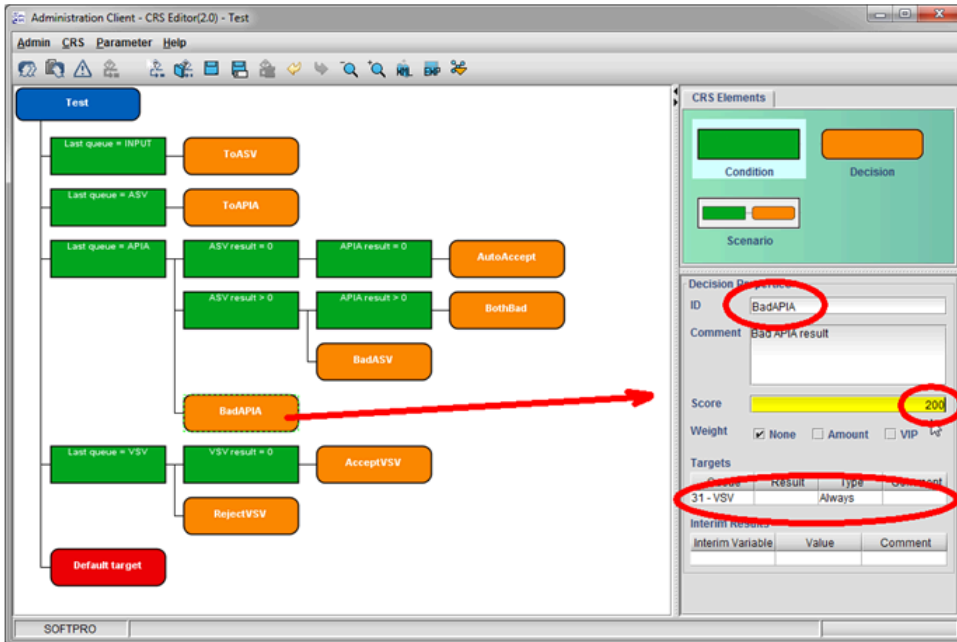
Items that reach the **BothBad** decision will now get a score of 800.

When you changed the target queue you have probably noticed the **Priority** field. Priority is what needs to be changed to affect the order in which items will be processed. If you leave the field empty, the rule will compute a new item priority based on the item risk (in this case the score). The higher the risk, or score, the lower the priority value (the lowest priority value gets processed first). If you enter something into the **Priority** field, you will override the system calculated priority. CRS will now use the value you entered, without calculating one based on the item score. For this example, leave that field empty.

You could now add two new scenarios, one that reads **ASV result > 0** and another one for **APIA result > 0**, or you can use the short cut built into the existing one. Items with bad ASV will pass the first condition, only items with both bad ASV and bad APIA will pass both. You just need to right-click on the **ASV result > 0** condition and select **Add decision** to catch items with bad ASV only. You will name the decision **BadASV** and give it a score of 600. Be sure to change the target queue to VSV.

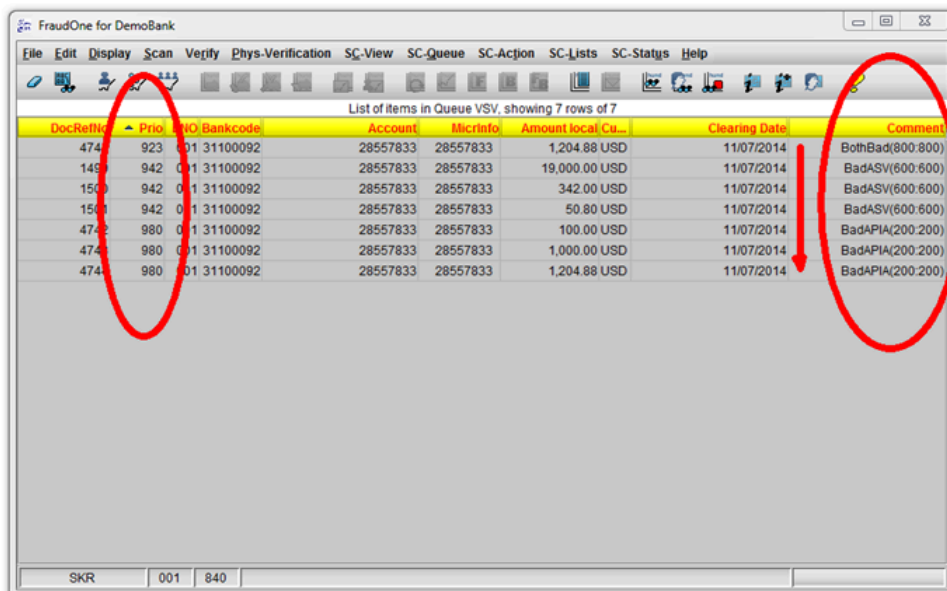


Now the only thing left is bad APIA. You caught both good results, both bad results and ASV only failure. Items reaching the **ToVSV** decision left below will be the ones having only bad APIA results. Change the decision accordingly.

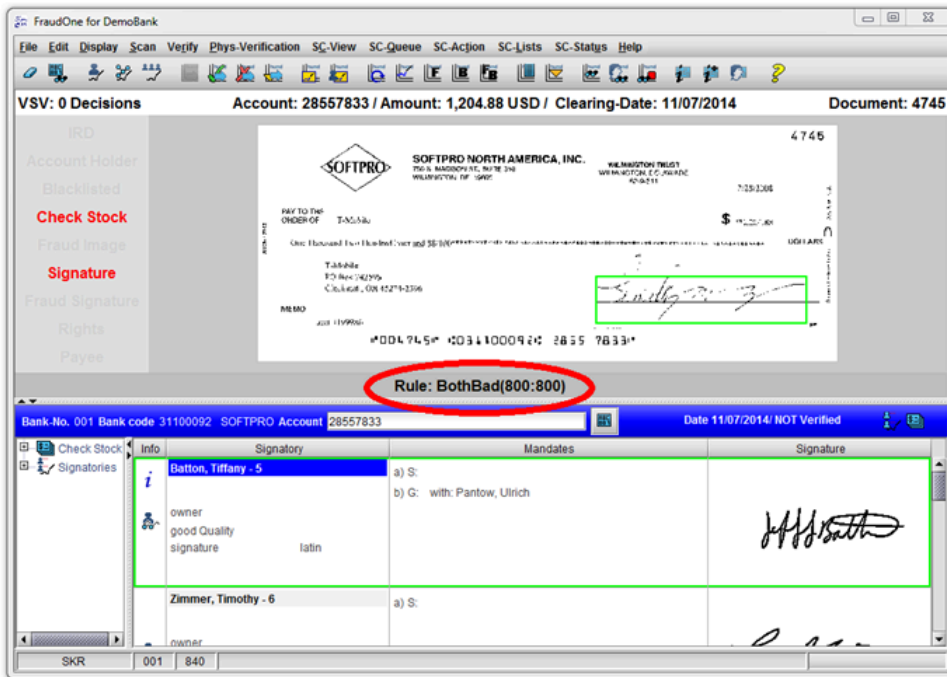


Name the decision **BadAPIA**, give it a score of 200 and move the items to VSV.

The rules are complete, you can try them out. Save the rule set. Stop all servers and restart them. Run the Getter and the verification engines. Now open the Java Client, get to the VSV queue and check the priority in the lists, or look at the order you get the items when you select **get next**. The ones with two bad results should be presented first. While you verify the item, notice that the name of the rule that applied is shown on the screen. That might help you when you decide the item.



Note that a lower number means a higher item priority.



The information line shows the rule name together with score and risk.

If you look at the priority ordering you can see that items with two bad results are shown first, bad ASV items second and bad APIA items last.

Don't forget to clean up the system when you are done (DFP).

How much does it hurt?

As discussed previously, the rule score only tells how probable it is that the item is fraudulent. However, you may want to treat items with the same score differently, depending on how important they are. A \$10,000 check will hurt more than a \$50. That is called item weight. Item risk is defined as:

$$\text{risk} = \text{score} * \text{weight}$$

CRS has three item weighting functions:

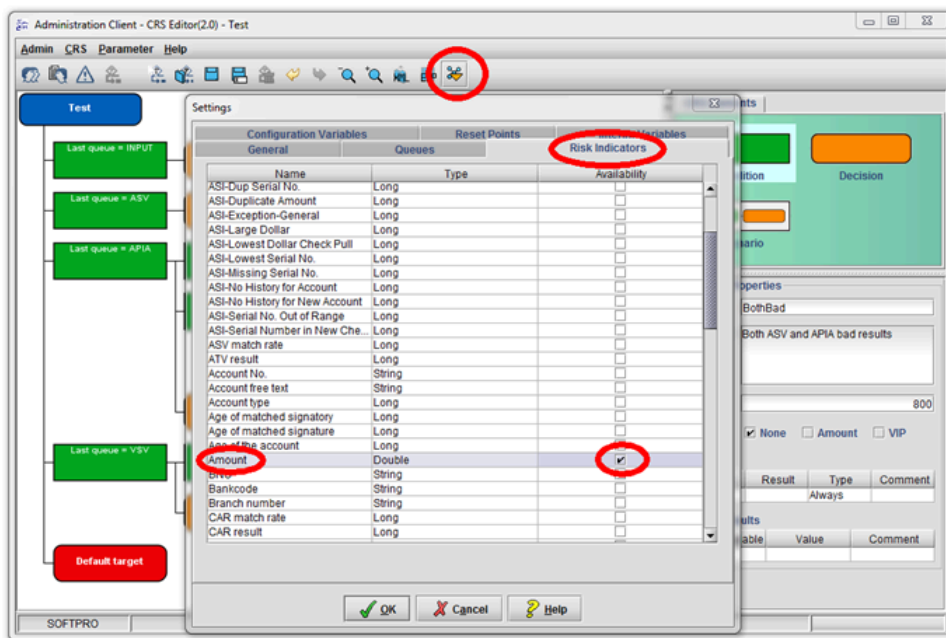
None	No weighting function. $\text{risk} = \text{score}$
Amount	The check amount is used as item weight. The higher the amount, the more important the item gets. $\text{risk} = \text{score} * \log(\text{amount} + 10)$

VIP	<p>Very important person (account). Accounts marked as important are given a higher risk than normal accounts.</p> $\text{risk} = \text{score} * \text{VIPmultiplier}$
-----	--

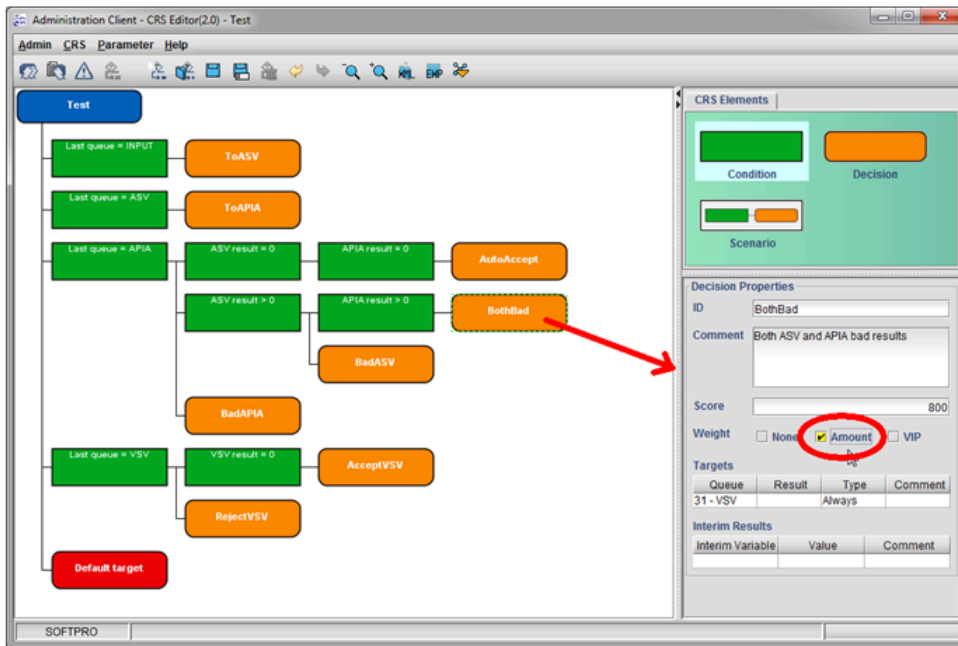
A combination on the above is possible. The general formula used to calculate risk is given below:

$$\text{risk} = \text{score} * \text{VIPmultiplier} * \log(\text{amount} + 10)$$

Open the CRS editor, select the **BothBad** decision and change the risk function to **Amount**. Before doing this, if you want to use amount, you will need to turn the amount risk indicator on. You cannot use risk indicators that are not enabled. Click the **Edit properties** button, go to the **Risk Indicator** tab and enable the **Amount** risk indicator:



Now you can go back to the **BothBad** decision and enable the **Amount** weight function:



Item risk will now be calculated from rule score and item amount.

Do the same for the other two decisions that select items for visual review: **BadASV** and **BadAPIA**.

Priority

But how will the risk information be used? The new item priority is calculated based on risk. Each time CRS moves one item from one queue to another, it gets a new priority. The higher the risk, the lower the priority will be. Items with highest risk (and thus lowest priority) will be processed first.

Run your items through the system again (don't forget to save and restart). You will find that, some items that hit only **BadASV** or **BadAPIA** now outrank items from **BothBad** if the item amount is high enough:

Table data from the screenshot:

DocRefId	Prio	SNO	Bankcode	Account	Micrinfo	Amount local Currency	Clearing Date	Comment
1463	753	01	31100092	28557833	28557833	19,000.00 USD	11/07/2014	BadASV(600.2567)
4775	762	01	31100092	28557833	28557833	1,204.88 USD	11/07/2014	BothBad(800.2467)
1590	853	01	31100092	28557833	28557833	342.00 USD	11/07/2014	BadASV(600.1527)
1501	897	01	31100092	28557833	28557833	50.80 USD	11/07/2014	BadASV(600.1070)
4774	940	01	31100092	28557833	28557833	1,204.88 USD	11/07/2014	BadAPIA(200.616)
4773	942	01	31100092	28557833	28557833	1,000.00 USD	11/07/2014	BadAPIA(200.600)
474	960	001	31100092	28557833	28557833	100.00 USD	11/07/2014	BadAPIA(200.408)

Notice the updated **Comment** field. The rule name is followed by score and risk. The risk is now different from the score and depends on the amount. Compare that output to the one where the risk ranking was done by score alone. Clean the system using DFP after you're done.

Wrap up

The rule graph determines the score of an item. The item score is determined by the decision that is selected from the rule graph. The score tells us how probable it is this item is fraudulent.

Each item has a weight. The weight tells us how much it will hurt us if this item is decided wrong. The weight of the item is determined by the weight function selected in the decision. It can be 1 (**None**), **Amount**, **VIP** or a combination of the two.

The risk of an item is the product of the score and weight. The higher the score and the higher the weight (amount) the higher the risk will be.

$$\text{risk} = \text{score} * \text{VIPmultiplier} * \log(\text{amount} + 10)$$

The item priority is computed based on the risk. The higher the risk, the lower the priority number will get. The lowest priority number will be processed first.

i Item priority is recalculated after each CRS pass (after each new result that becomes available). Both engines and visual clients will process items by priority, thus the most important items will be the first to be displayed.

Chapter 6

Managing volume

Risk based decisions

Previous chapter you managed to sort items by priority and process the important ones first. That, however, does not help you to deal with changing daily volumes. Even if you fine tune the system for an average day, there are still weekdays (Monday?) or dates around holidays where you will get a significantly higher input volume.

The one thing you could do is, change your rules every Monday so that fewer items make it into your review queue. Then change them back on Tuesday. Actually, there even is a way to do and automate this. On the other hand, CRS provides for an easier option.

i You can save two different rule configurations and then have the scheduler component set up a timetable and start your system using different configurations on different days. See *Kofax FraudOne Administrator's Guide*.

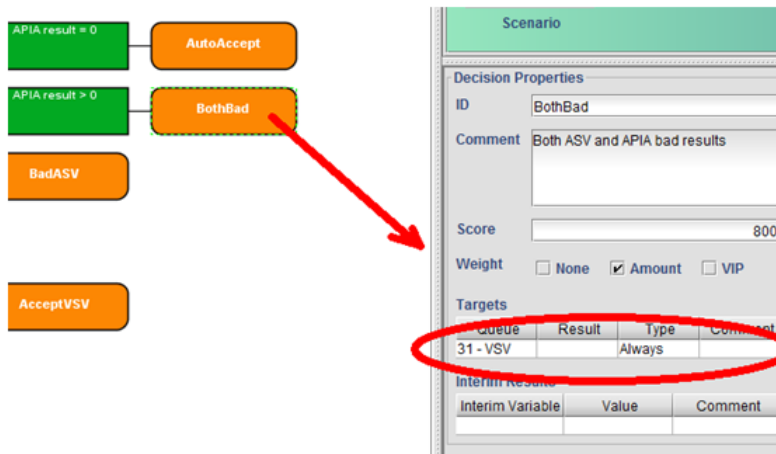
You ranked the items by risk (see previous chapter). The higher the risk, the more important the item is. The only thing you need to do is, treat items with high risk different from items with low risk.

Global threshold

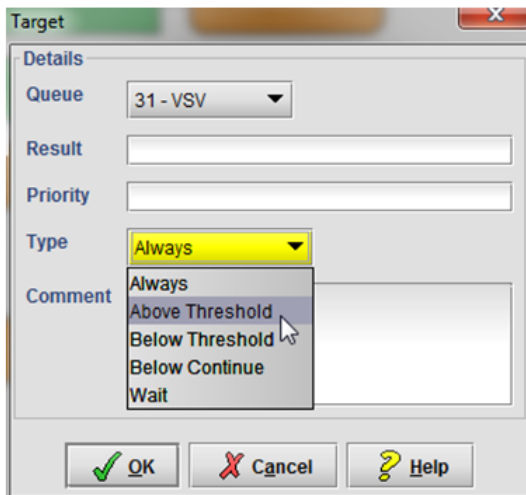
CRS defines a "global risk threshold" and a way of handling items above the threshold different than the ones below. You can now adjust the global threshold only, and have volume management by the turn of a button.

Open the CRS editor again and load the work you saved during the previous chapter. What you want to do is to limit the number of items that make it to the VSV queue. Only items with a high risk should be presented for review, items with low risk will be sent directly to the OUTPUT queue to be paid.

Three rules (decisions) are responsible for putting items into VSV: **BothBad**, **BadASV** and **BadAPIA**. Those are the ones you will need to change. Pick **BothBad** first and look at the target definitions. There is only one and it says "put items into the VSV queue always":

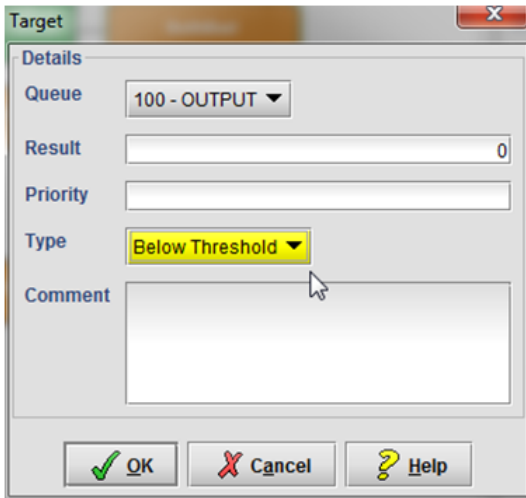


What you want to change is, only put items into VSV above the risk threshold, not always. You will need to edit the target. Right-click on it and select **Edit**. The target properties will be shown. Change the target type to **Above threshold**:



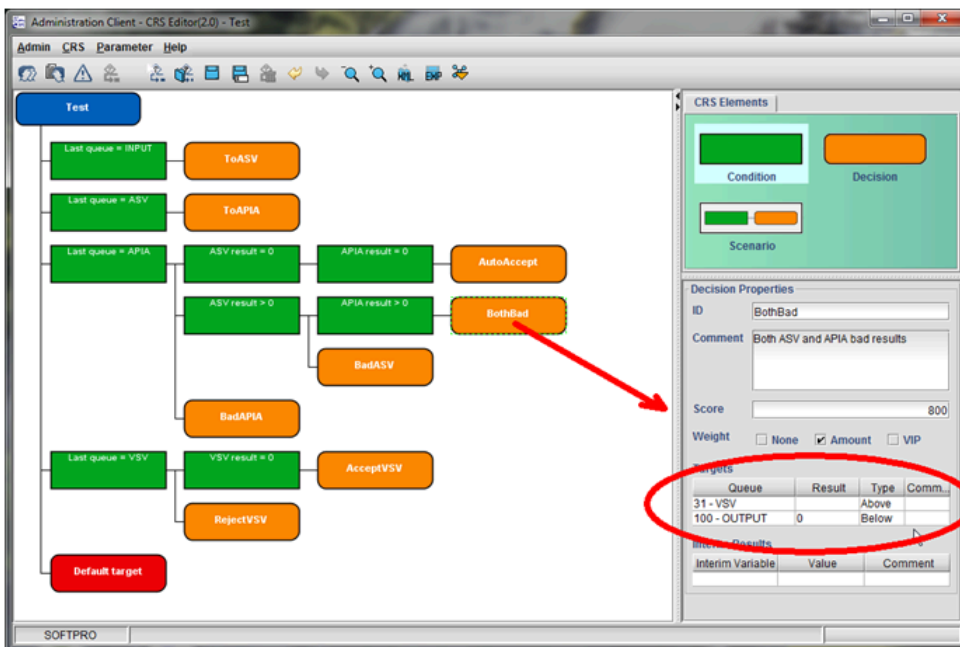
Now this target only applies to items where the individual item risk is above the global threshold. Only high risk items will be sent to VSV.

But what happens to items with a risk below the threshold? They would not be sent anywhere (the default error handling target would apply). You need to take care of them. Right-click on the VSV target and select **New target**. A new target will be created. You need to decide then what to do with low risk items, such as to auto pay them. Set the queue to OUTPUT and put 0 (ok) into the result field. Set the type of the target to **Below threshold**:



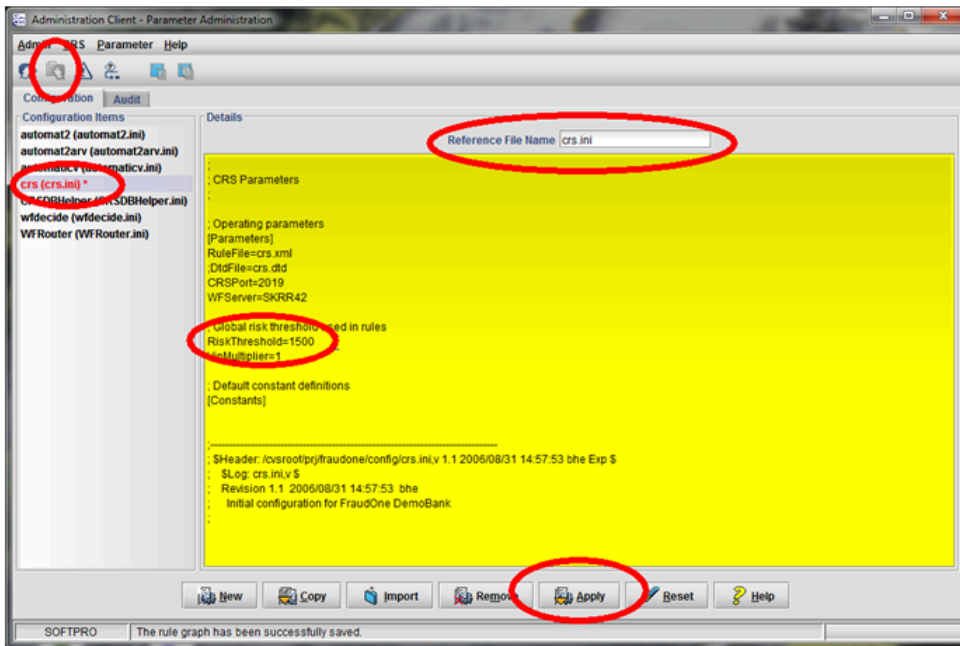
This target will now handle all items with a risk below the threshold. These items will be sent to OUTPUT with a result of 0 (pay).

The decision you changed now has two targets – one applies to items above the threshold and sends the items to manual review, the other one applies to checks below the threshold and sends them to OUTPUT with a 'pay' recommendation:



Change this for the other two decisions (**BadASV** and **BadAPIA**) as well and save the rule set.

Now that everything is in place, you just need to set the initial global threshold to be able to try this out. Change to the parameter administration section in the Administration Client and select the configuration item that has crs.ini as a file name.



Locate the **RiskThreshold** item in the **Parameters** section (or create one if it does not exist). Now set a value. You'll need to check the risk of your own items and calculate a suitable value. For this example you can look at the table you generated in the previous chapter:

The screenshot shows the 'FraudOne for DemoBank' interface. The table below shows a list of items in the queue. The 'Risk' column is highlighted in red, and a red arrow points to the 'Clearing Date' column.

DocRefId	Prio	RNO	Bankcode	Account	MicroInfo	Amount local	Currency	Clearing Date	Comment
145	753	01	31100092	28557833	28557833	19,000.00	USD	11/07/2014	BadASV(600.2567)
475	762	001	31100092	28557833	28557833	1,204.88	USD	11/07/2014	BothBad(800.2467)
1500	853	001	31100092	28557833	28557833	342.00	USD	11/07/2014	BadASV(600.1527)
1501	897	001	31100092	28557833	28557833	50.80	USD	11/07/2014	BadASV(600.1070)
474	940	001	31100092	28557833	28557833	1,204.88	USD	11/07/2014	BadAPIA(200.616)
473	942	001	31100092	28557833	28557833	1,000.00	USD	11/07/2014	BadAPIA(200.600)
474	960	001	31100092	28557833	28557833	100.00	USD	11/07/2014	BadAPIA(200.408)

The risk numbers are on the right. '1500' seems a suitable number for a test. It should cut out the lowest four items.

Set the **RiskThreshold** to 1500 then and click the **Apply** button. All you now need to do is restart the system and feed the items back in using the Getter. If you now look at the VSV queue again you will find it considerably shorter:

DocRefNo	Prio	BNO Bankcode	Account	MicrInfo	Amount local Cu...	Clearing Date	Comment
1499	753	001 31100092	28557833	28557833	19,000.00 USD	11/07/2014	BadASV(600:2567)
4745	762	001 31100092	28557833	28557833	1,204.88 USD	11/07/2014	BothBad(800:2467)
1500	853	001 31100092	28557833	28557833	342.00 USD	11/07/2014	BadASV(600:1527)

The items with a risk below 1500 are now missing from the list. They have been moved to the OUTPUT queue instead.

By changing the one global risk threshold you can manage throughput in all queues across all rules. You now have one single parameter that can be used to manage check volumes for the whole system. No need to change all rules (three in our case, but probably much more in a real world system) to adjust for volume changes.

You can change the threshold manually, as shown, or use the Scheduler component (your system administrator can help with this process) to set up a time table specifying when to use which setting.

When you're done, you can use DFP to clean the system.


i If you use a target **Above threshold** do not forget to handle items below the threshold too! **Above threshold** and **Below threshold** should be used in pairs.

Continue?

While you changed the target type you noticed another one: **Below continue**. Here is what that can be used for:

When you set up our rules and calculated risk, you found that you need to put rules with high scores first, because the system will stop at the first decision that applies. If you now introduce the global risk threshold, you can handle items above and below it differently. You can choose to

present items above the threshold for review, but you are also forced to do something with items below it. What if you don't want to decide them yet? That is what the **Below continue** target does. It basically tells the system to ignore this decision and continue parsing the rule three as if it would not exist. The parser will try to find the next decision that applies and execute it.


 Be careful when using this target type. You'll get an error if no other decision applies and rules using it tend to be a bit confusing.

Wrap up

FraudOne includes an easy mechanism for volume management. Item risk can be used to select different targets out of the decision that has been reached.

Targets of the type **above threshold** apply to items with a risk above a configured threshold and targets of type **below threshold** apply to items below it. Thus different targets can be chosen depending on item risk.

By changing one global threshold items can be moved one way or another. No need to adjust all rules.

 You need to use 'above' and 'below' carefully in order not to miss items (best used in pairs).

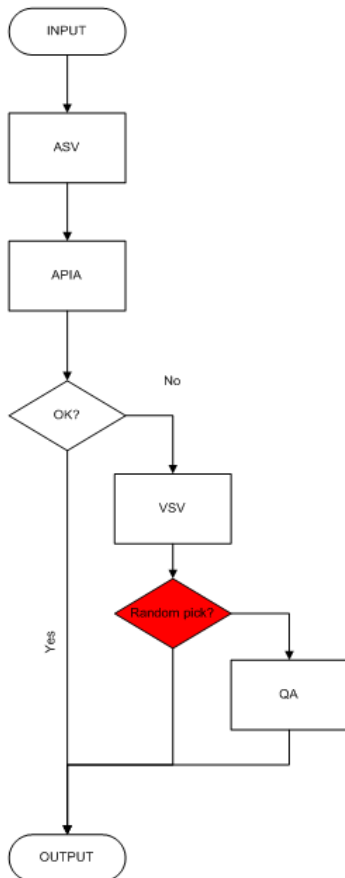
Chapter 7

Random decisions?

When you don't want to control everything

What good is it to make random decisions? Consider following example: you have a QA department that needs to verify everything is working correctly. They could randomly pick some checks and verify they were processed the way they were supposed to.

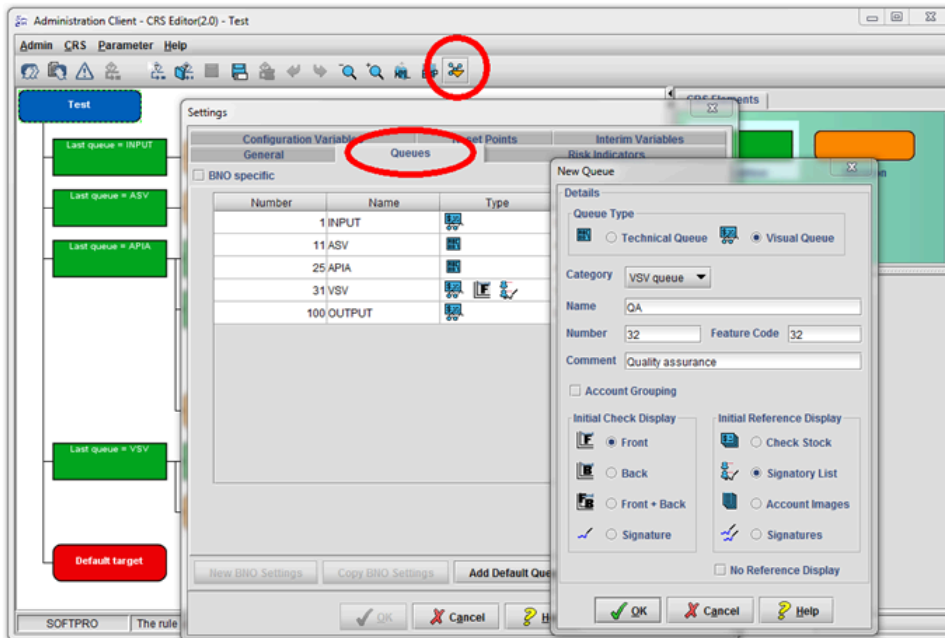
What you need to do is, set up a new queue you will name **QA** and randomly move some VSV decisions there to be verified:



Define a new visual verification step

Start your CRS editor and load the rule set. Open the parameters window by clicking the **Edit parameters** button. Change to the **Queues** tab. You will now see the list of currently used queues.

Add another one by right-clicking into the queue list and selecting **Add**.



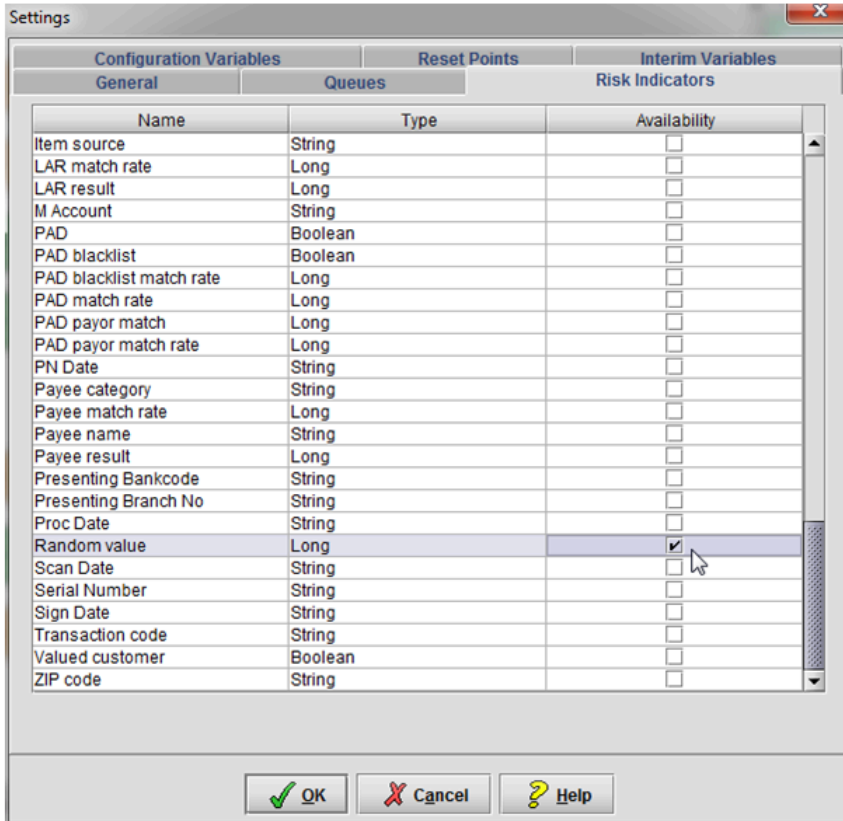
Change the type of the queue to **Visual queue**. That is the one interactive clients work on, as opposed to **Technical queue** used by automated engines. Set the category to **VSV queue**, name it **QA** and give it a number of **32**. Use this as a feature code too. The initial check display should be set to **Front**. You can decide if you want to see the signatory list or the check stock reference first.

Click **OK** after you're done. The queue will now show in the list and can be used in the rules and workflow. On the other hand, there will be users working on that queue and those users will also make decisions. Those decisions will show as the **QA** risk indicator. In order to use it, you will need to enable it. Move to the **Risk indicators** tab, scroll down to the **QA result** indicator and enable it.

Use a random risk variable

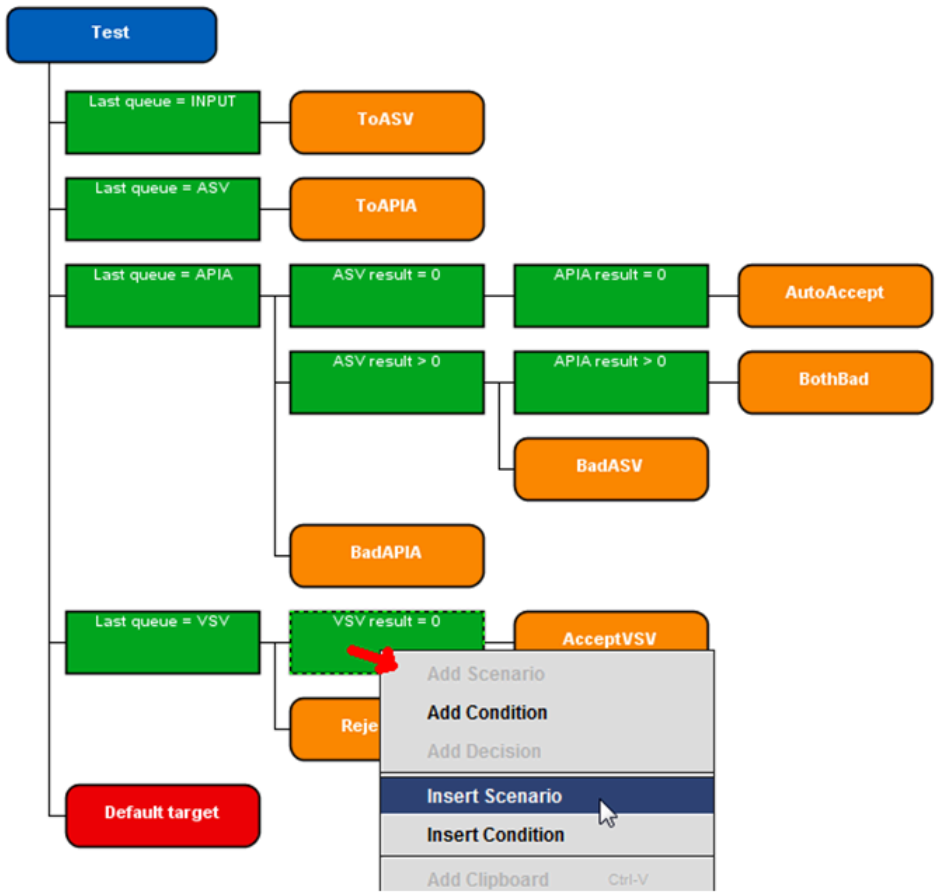
Now that you have the new queue, you can set it up to receive some of the items decided by VSV for a review, say a random pick of 15%.

First, you need to enable a random risk indicator. In the parameters window move to the 'Risk indicators' window and search for an indicator named **Random value** and enable it.

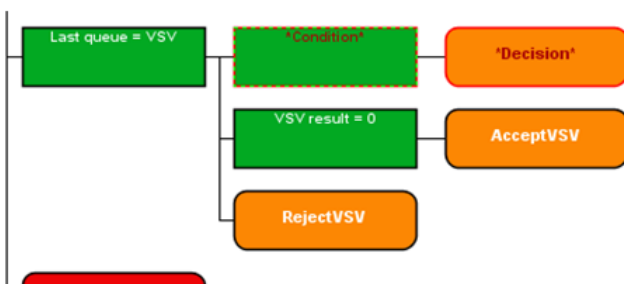


Close the parameter window by clicking **OK**.

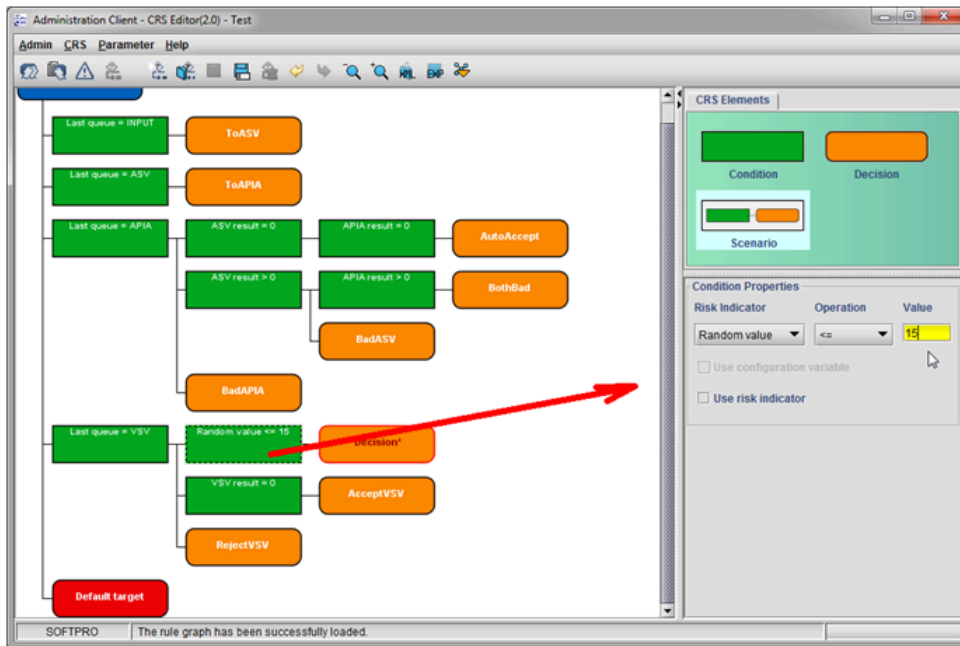
Now identify the spot in the rule graph where VSV items are processed. You'll find it in the branch named **Last queue = VSV**. Before you even check the VSV result, you want to pick 15% of the items at random. Right-click on the **VSV result = 0** condition and select **Insert scenario**.



A new scenario will be inserted right above the condition you clicked on:

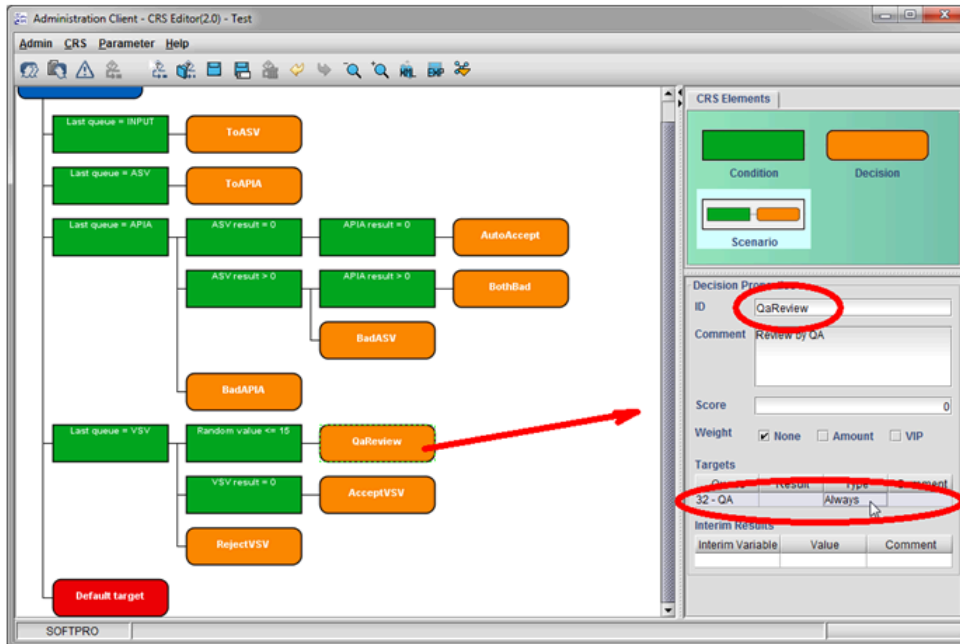


Now click on the new condition. Change the risk indicator to **Random value** which you just enabled, set the operation to **<=** and the value to **15**.



The condition now uses the random value risk indicator. This indicator will have a random value between 0 and 100. Comparing it to less than 15 will mean the condition is true for about 15% of the time. Thus about 15% of the items will pass it, while the rest will be processed by the condition and decisions below, as before. If you want to define different ranges, you can create your own random values by setting up custom risk indicators. More on that will follow later.

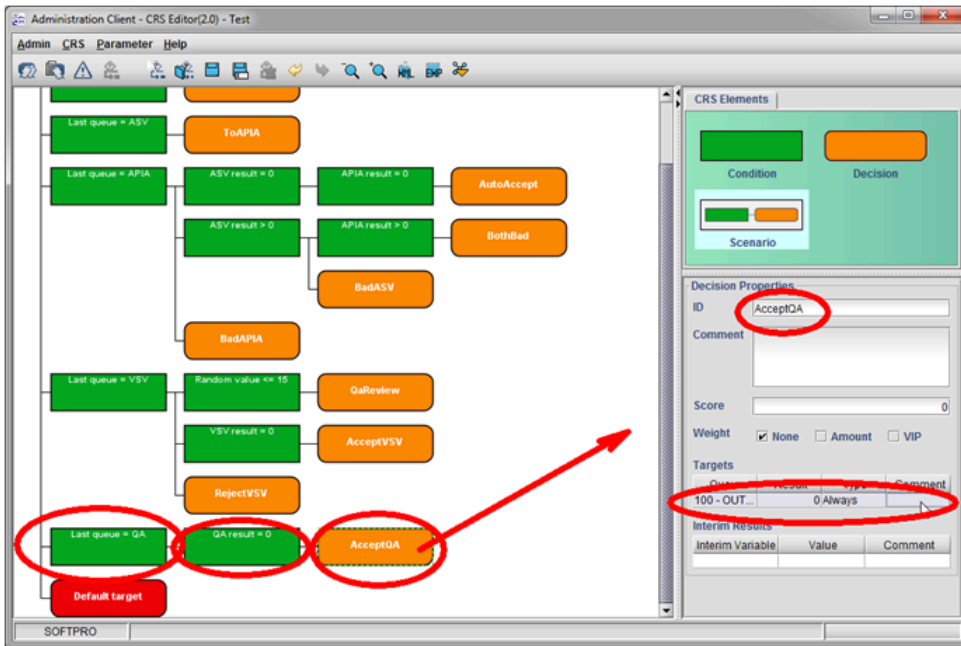
You will now change the decision following our random condition to move the selected random pick to the **QA** queue. Change the name to **QaReview** and edit the target in the list. Set the queue to **32 - QA** and the type to **Always**.



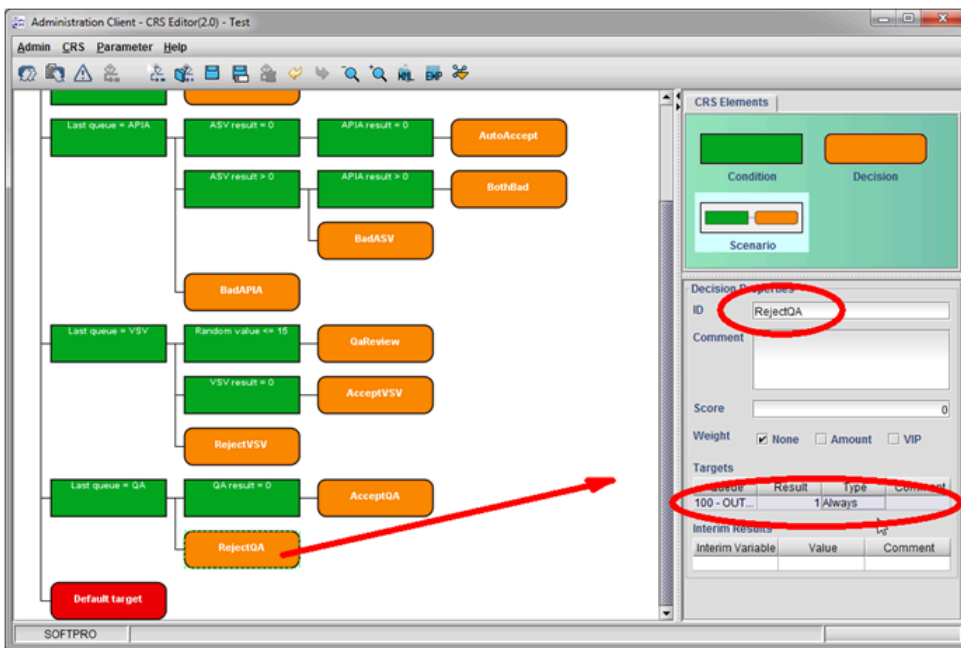
Let us have a look at the VSV items. They are the ones that pass the **Last queue = VSV** condition. The first condition they encounter is **Random value <= 15**. Each item will get a different one, in the range of 0 to 100. For about 15% of them the condition will be true, they will pass it and be sent to **QaReview**. The others will try the next condition down the line, which is **VSV result = 0**. If that is true, they will be sent to OUTPUT with a result of 0 (the **AcceptVSV** decision). If not, they will hit the decision **RejectVSV** and be sent to OUTPUT with a result of 1.

You have sent around 15% of the VSV items to **QA**. Next you need to tell the system what to do with items coming out of **QA**.

Right-click on the **Test** rule tree base and select **Add scenario**. A new rule branch will show on the graph. Change the first condition to **Last queue = QA**. It now applies only to QA items. What do you need to do with them? You will treat them the same as VSV items. All will move to OUTPUT, accepted ones with a **0** result, all others with a **1** result. Right-click on **Last queue = QA** and add a condition. Change it to **QA result = 0**. Change the decision behind it. Name it **AcceptQA**. Change the target in the list setting **Queue** to **OUTPUT**, **Type** to **Always**, and **Result** to **0**.



That has taken care of the accepted items. All others you can grab by branching a new decision before the **QA result = 0** condition. Right-click on **Last queue = QA** and select **Add decision**. A new decision will show after it and below **QA result = 0**. Change the ID to **RejectQA** and edit the target in the list. Set the queue to **OUTPUT**, the result to **1** and the type to **Always**.



That's it. The **QA** queue is now in place. If you want to test it, save your work, then restart the servers. Load some items using the Getter and use the Java Client to make some decisions. Some,

but not all of those items will make it to the new **QA** queue. You might need a bigger number of checks to verify this due to the random selection process.

If you want to use the new **QA** queue, you might need to change the user rights to be able to access it.

Wrap up

You can use the CRS editor to create new, custom, queues. These will be either technical queues (used by automated engines) or visual queues for user review. These newly created queues are immediately available in the rule design.

To facilitate random decisions, or random picks, the CRS supports a random risk indicator. It has a range of 0 to 100 and easily translates into a % value.

Chapter 8

Planning for change

Change management is something you should think about. Consider a medium sized rule set with tenths or hundreds of elements. The conditions will contain a lot of values to compare against, some of them being related. Each time you have to change one of those values, you need to edit the condition inside the rule graph and save a new copy of the rules. If there is a dependency between two values in the graph, you need to bear that in mind and manually change both.

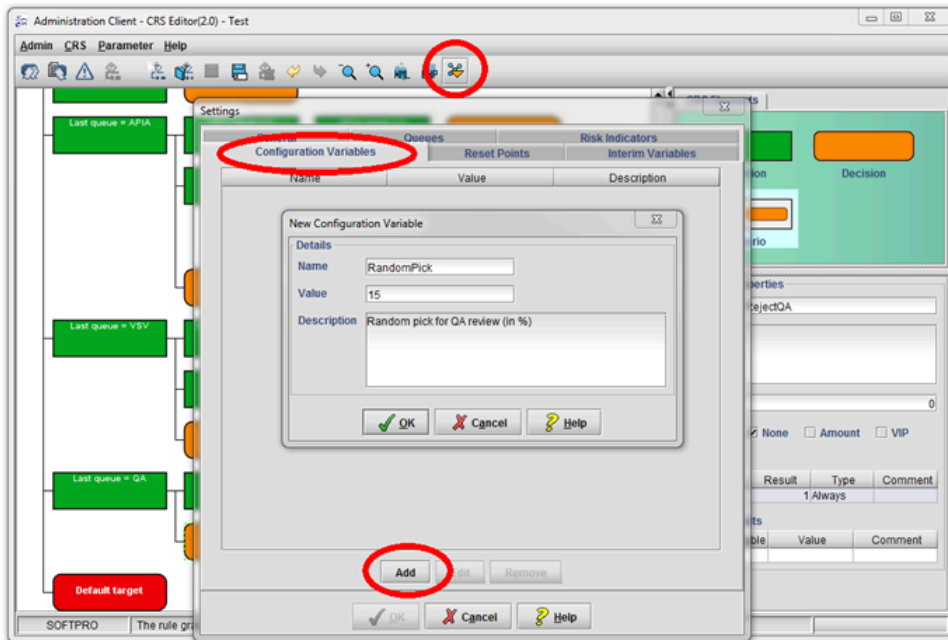
A better way to do that is to use configuration variables.

Configuration variables

Configuration variables replace the value inside the condition with a symbolic reference. You can assign that reference a value inside the CRS parameter file and do not need to edit the rule graph each time. Also, if two condition values are related, you can assign both the same symbolic reference and do not need to care about the link henceforth.

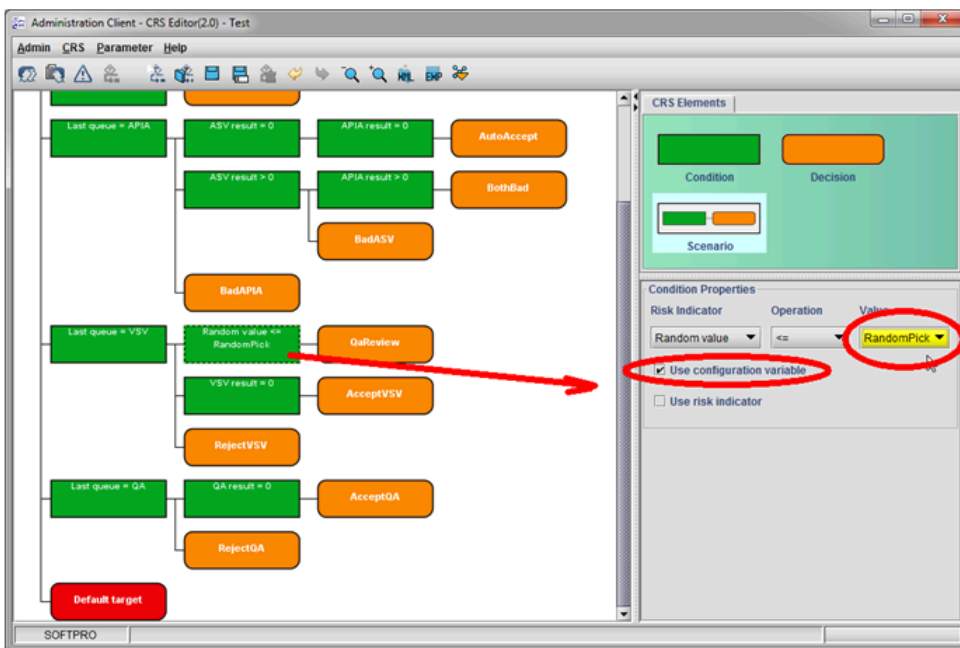
Here is an example that changes the 15% random value defined in the previous chapter to a variable.

Open the CRS editor, load the rule set and open the parameter window. Move to the **Configuration variables** tab. You will see an empty window. Click the **Add** button at the bottom and the dialog for a new variable will open:



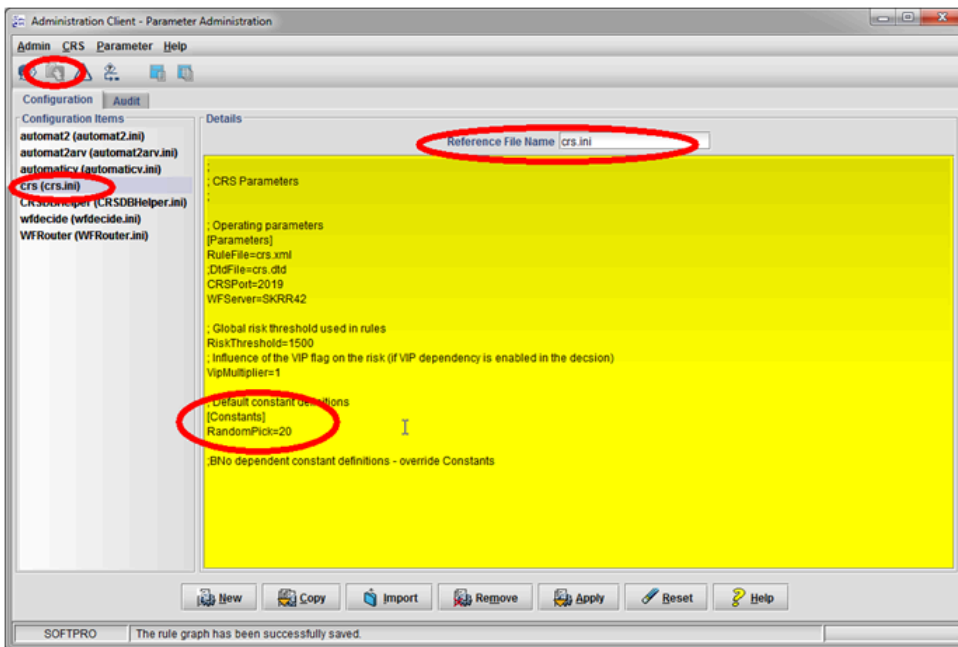
You will give the variable the name **RandomPick** and put a meaningful comment into the description field. The setting in the **Value** field is the default value. That value will be used if no other has been specified in the configuration.

Now close the parameters window and select the **Random value <= 15** condition from the rule graph. In the condition properties check the **Use configuration variable** check box.



Now you can select the configuration variable instead of the value entry field. Pick the one you just defined. Save the rule set. You can now use the new variable.

The question now left is: how do you change the assigned value? While in the Administration Client, change from the CRS editor to the parameter maintenance. Select the configuration file for CRS (usually named crs.ini). Inside the text bloc you will find a section named **[Constants]**. Here is where you enter our definitions. Define the value **RandomPick=20** as shown below:



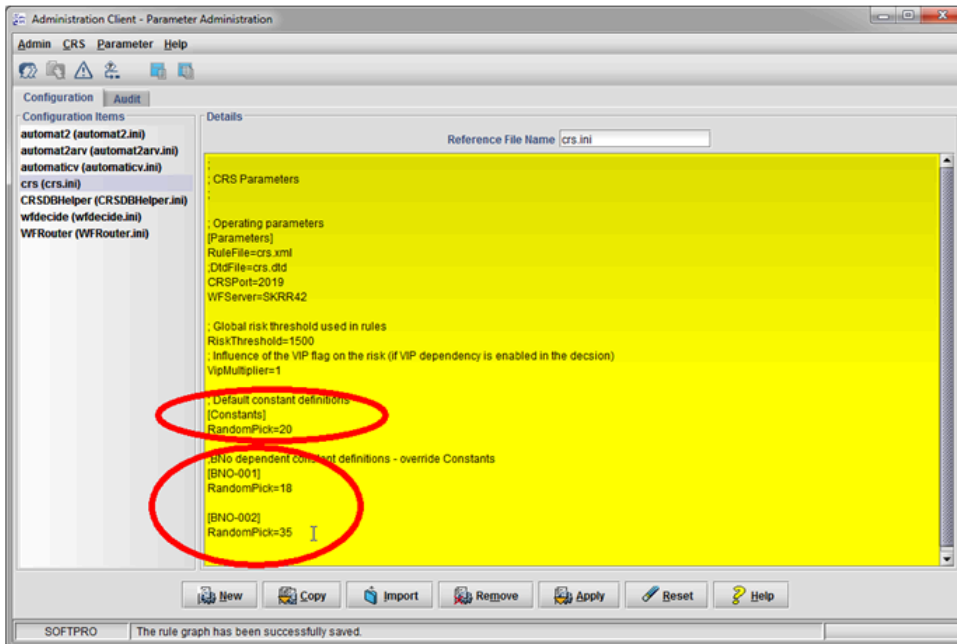
Apply the changes and restart the servers. CRS will now use the value 20 instead of 15.

The advantage about this is, that:

- You don't need to change the rules each time you want a value adjusted.
- Dependent conditions and values can be treated as one variable by using the same variable inside multiple conditions.
- By carefully assigning user rights you can have a group of users change specific values without allowing them to edit the CRS rules.

BNO dependent variables

You can set different values for different BNO's without editing the rule set. Go to the configuration file again and add two new sections as shown below:



Apply and restart. What the CRS engine will do now is, use the value '18' for checks belonging to BNO '001', '35' for checks belonging to BNO '002' and '20' for the rest. You don't need to set up a different rule for each BNO.

To summarize:

- You define a configuration variable in CRS and use it inside rule conditions.
- The actual value used will be:
 - The value defined under 'BNO-xxx' for the actual check BNO.
 - If the above is not set, the value defined under 'Constants'.
 - If that too is missing, the value defined as default in the CRS editor.

Wrap up

Configuration variables can be used instead of values inside condition objects. That allows the value to be defined outside the CRS rule set.

The variable value will be defined in the CRS configuration via Configuration Server. You can change values without the need to edit the rule set. You can have different persons change the variables and others edit the rules.

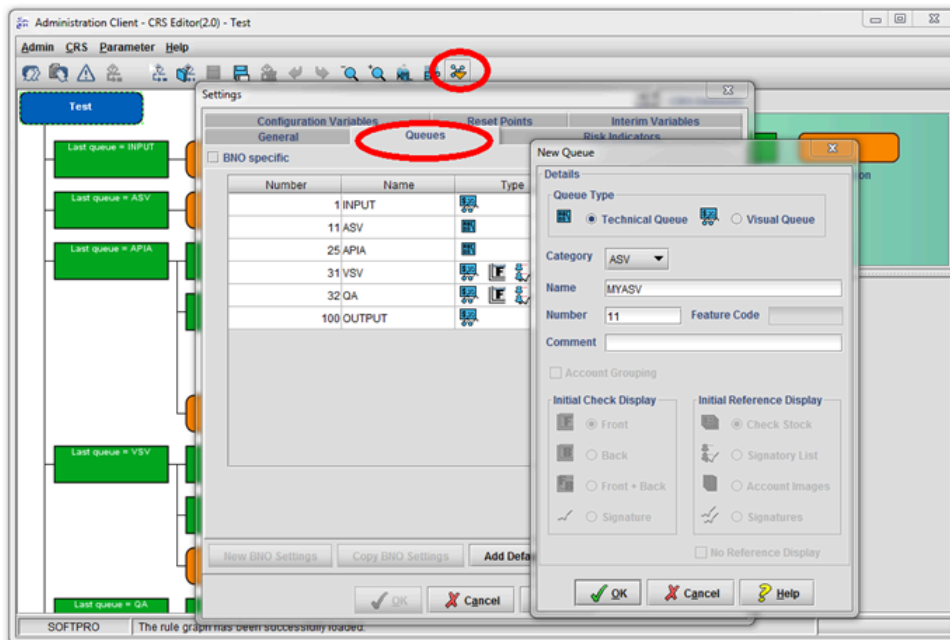
Configuration variables can be BNO dependent. You do not need to write BNO dependent rules to have BNO dependent conditions.

Chapter 9

Add your own queues, engines and indicators

Add a custom queue

Adding a new queue is simple. Go to the properties window and select the **Queues** tab. Now right-click inside the list of queues and select **Add**. The configuration for the new queue will show:



Queues can be:

- **Technical**, or non-visual. These queues are used for automated verification like ASV, APIA (check stock).
- **Visual**, or manual. These queues are used by visual clients like Java or Thin Client. Human users are presented items for review and decision.

If you want to add your own engine, you will pick a technical queue type. FraudOne engines have predefined queue numbers. To add an external (non FraudOne) engine, set the category to

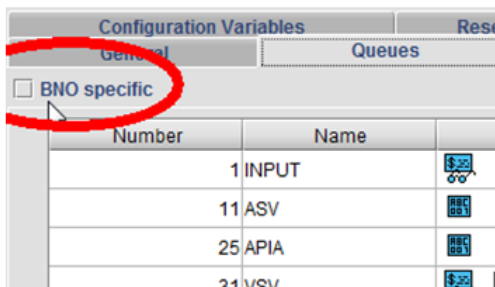
External and assign the engine a queue number. Make sure the engine actually uses the same queue number that you just entered into the queue description.

If you want to add a new visual processing step, you will pick a visual queue type. Now you can also set the display parameters for the clients that will work this queue. Java and Thin clients will use the CRS rules configuration to determine how to initially show item and reference data. Once you defined the new visual queue, the associated risk indicator will become available under the name **<queue name> result**. In order to use it in the CRS rules, you will need to enable it in the **Risk Indicators** tab.

BNO specific queues

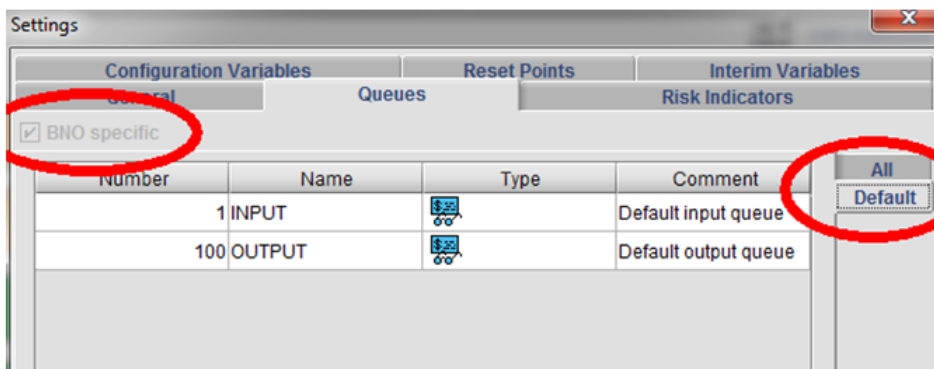
Some FraudOne installations use a BNO dependent workflow. Most often this happens if a processing center works items for different mandates, but some banks also use a different setup for different internal organizations or branches. CRS supports this scenario too.

If you want to use BNO dependent queues, you need to select the **BNO specific** check box below.



Use a BNO specific queue setup only if needed, that is, if your clients need to show a different list of queues, or use different queue names, for different BNO's. Most of the CRS rule sets will not need it. Be warned! Once set, there is no way back. Therefore, if you want to experiment using it better create a new set of rules if you want to keep the one used for your other experiments so far.

Once you select the **BNO specific** check box, you will notice that a new tab has appeared at the right of your queue list.



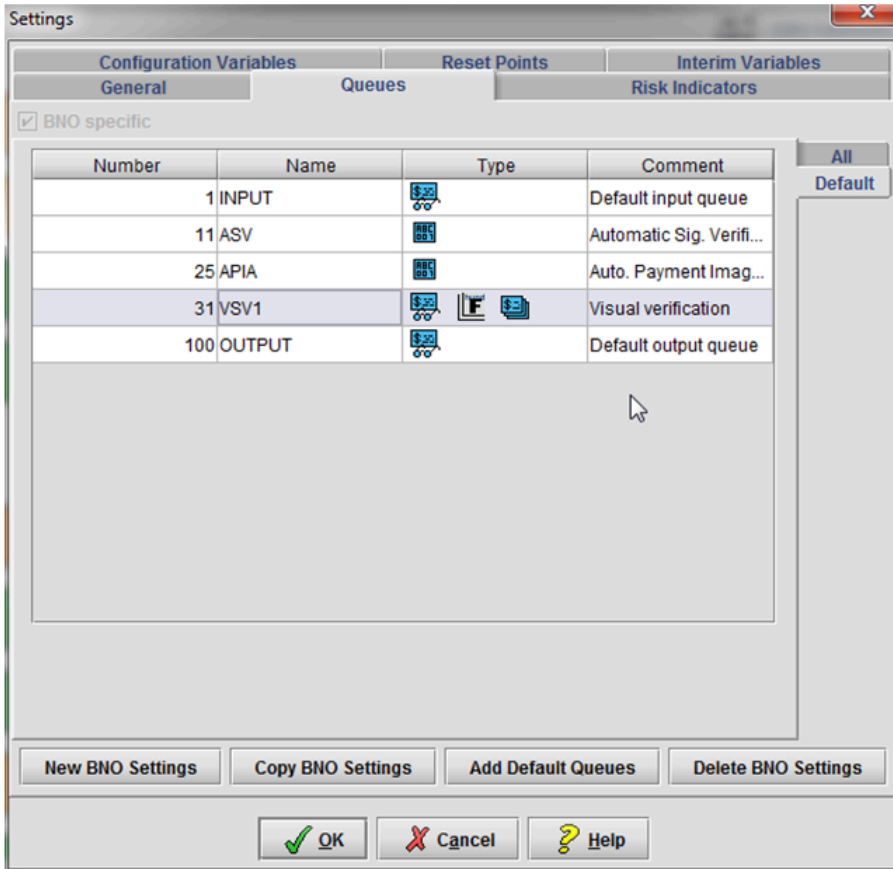
Each tab contains a BNO list. The meanings are:

<number>	Defines an individual BNO. If a visual client works with this BNO, it will show only the queues listed in this tab.
Default	Queues defined in this tab will be used for all BNOs that have not been individually specified.
All	List of all queues used in all BNOs. This is what the non-visual components will use. Also CRS will rely on the names in this list when providing queue identification. Visual clients do not use this list except when working the pseudo-BNO '999'. If you add a queue to a specific BNO, it will also be added to the All list automatically.

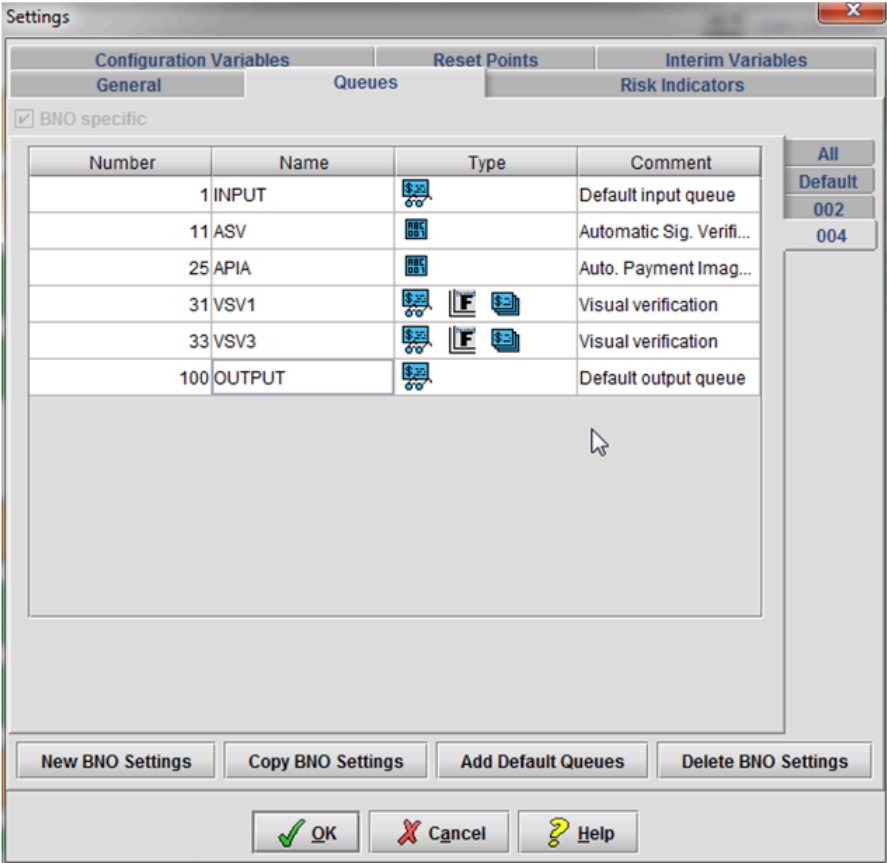
Here is a specification you can try to define in the system. Assume we use four BNOs with this list of queues:

001	002	003	004
<ul style="list-style-type: none"> • INPUT • ASV • APIA • VSV1 • OUTPUT 	<ul style="list-style-type: none"> • INPUT • ASV • APIA • VSV1 • VSV2 • OUTPUT 	<ul style="list-style-type: none"> • INPUT • ASV • APIA • VSV1 • OUTPUT 	<ul style="list-style-type: none"> • INPUT • ASV • APIA • VSV1 • VSV3 • OUTPUT

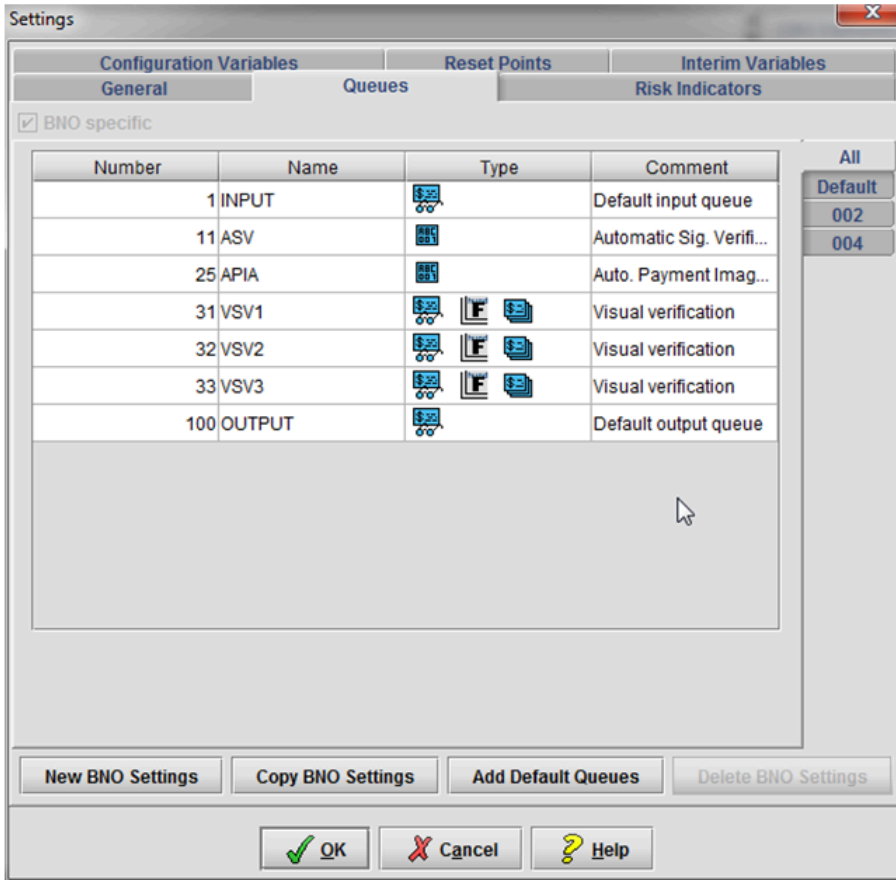
Start by defining the **Default** setup. That is what clients will use if no other BNO specific definition overrides it. Notice that BNOs '001' and '003' have the same list. You can use that as the default. Add the queues to the list the same way you did it before, creating the queues INPUT, ASV, APIA, VSV1 and OUTPUT.



Now check if you need to define any BNOs that have a different queue list than the **Default**. You should have, else there is no point in using a BNO dependent setup. In our example, that will be '002' and '004'. Define each one of them individually, by clicking the **New BNO settings** button and creating a new tab for that BNO. Add the queues shown in the table above.



Notice that the **All** tab shows the sum of all queues added so far.



Some points before we complete this setup:

- The first consideration for the **Default** setup should be what the client should show when new BNOs are added to the system, but no definition has yet been made. Only if you are fairly certain, that no default for new BNOs is needed, you can use the default to cover the most common BNO setup.
- Queues must not be named the same way in all BNOs. It is the queue number that defines it to the system. When editing CRS rules the names defined in the 'All' tab will be used to identify the queues.
- When you delete queues from one BNO, or a complete BNO at once, you should consider if the queues in the **All** tab used by the non-visual engines are still needed. You can only delete queues from the **All** tab that are not used in any of the other BNO tabs.
- You can add a queue to the **All** tab, without adding it to any of the BNO tabs, although this setup is not very common. What will happen is, the queue will not be visible in the visual clients while they work a specific BNO. You can use this to hide technical queues you don't want the client to display.

The way a visual client finds the list of queues to use is:

1. The client checks if a queue list has been provided for the specific BNO it is working on. If yes, that exact list will be used.

2. If no list has been provided for the specific BNO, the client will use the **Default** list.
3. If a client is working on all BNOs (pseudo BNO '999') the **All** list will be used.

Define your own risk indicators

As shown above, if you define a new visual queue, the associated risk indicator (queue result) will become available automatically. In addition to that, the system will supply the most commonly used indicators by default. You just need to enable them.

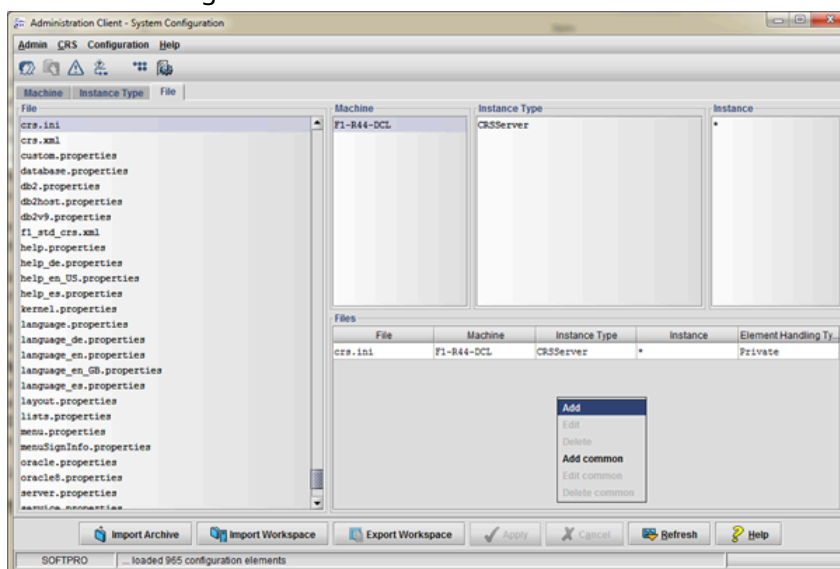
On the other hand, the FraudOne system stores a lot of other information that can be useful when deciding an item. Also, there is the ability to factor in data or results from external sources (like ASI 16, etc.). Depending on the location of the data, you have two ways to make it accessible. If the data is located in one of the standard FraudOne database tables (and extension tables to the SignBase reference data, or SignCheck results table) you can simply configure a new risk indicator yourself. If the data is stored in a non-standard location (customer specific tables or interfaces to other systems), a customer specific extension plug in that will make that data available as a risk indicator to CRS.

In the second case, contact your Kofax sales representative or technical contact person. This chapter will show you how to deal with the first case.

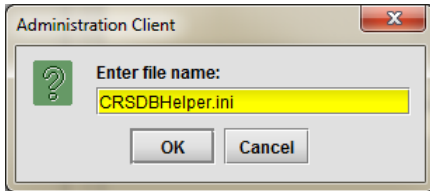
Since it is depending on the system you are currently using to perform the tests, we will redefine the VSV result we used above.

Open the **System Configuration** section of the Administration Client. Initially the parameter or configuration files panel is opened. Here you can create and change configuration elements for the FraudOne components. Risk indicators are defined in a configuration item of type file named CRSDHelper.ini. If none is present yet, you can create a new private configuration item as follows:

1. Add a new private configuration element of type file via opening the context menu in the **Files** list and selecting **Add**.

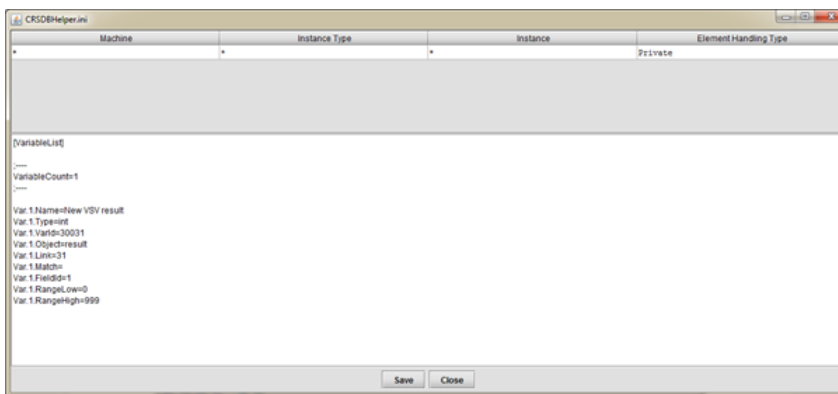


2. Enter the file name in the opened dialog and click **OK**.



i It is important that the file name is called CRSDBHelper.ini. Type in the name exactly. The system is case sensitive, which accounts for about 80% of configuration errors.

3. Copy or type the content below into the opened configuration editor window and click the **Save** button.



The section name `[VariableList]` tells the DB interface it finds the configurable risk indicators here. `VariableCount` specifies how many indicators you want to configure. For this example, you are only doing one, so set this to 1.

Each variable defined starts with `Var.n.`, where `n` is the variable number. The first will use 'Var.1.', the second 'Var.2.' and so forth.

If you want to check all options, open the *Kofax FraudOne Administrator's Guide* and look this section up.

The Name parameter will determine how the indicator will be shown in the risk indicator list. For this example, it will be named **New VSV result**.

The Type parameter determines what type the variable will have. The VSV result will be stored in the SC_RESULT table and thus will be an integer (int).

⚠ Selecting the wrong type might cause your system to behave in an undetermined fashion and in the worst case to crash. Check the database content of what you want to access, read the reference and carefully select the correct type.

The `VarId` parameter specifies the feature code used to identify the data. A complete list is found in the *Kofax FraudOne Administrator's Guide*. User defined variables will be in the 30000 range. Remember, in the previous chapter you defined the VSV feature code to be 31? This is why this entry will read $30000 + 31 = 30031$.

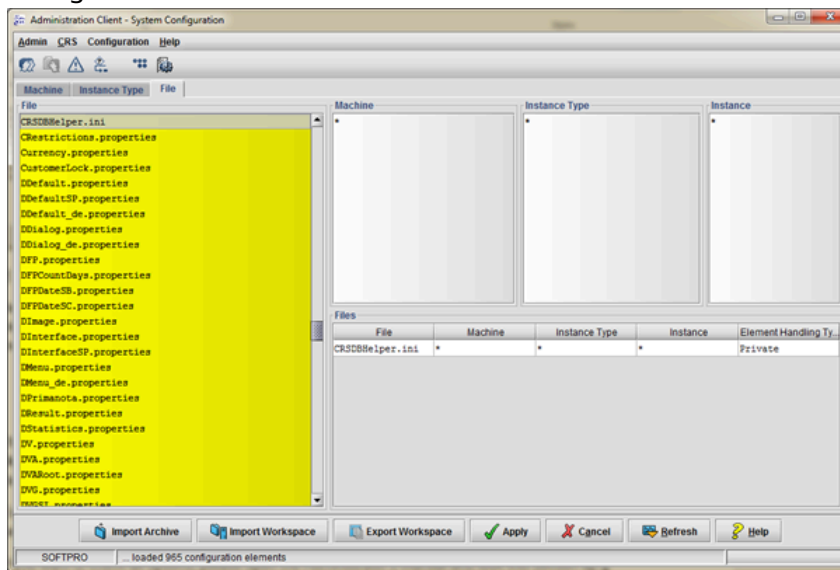
For the result, you should also set `Link` to match the feature code (31).

Match and FieldId tell the system which part of the table to use. For the result section in SC_RESULT set the FieldId to 1.

The range parameters (RangeLow and RangeHigh) tell the editor what range the indicator is allowed to have. The editor will not accept an entry outside this range.

The definition is now complete. You just need to save it and restart the systems. Since the Configuration Server reads this information only on system start up, a restart is needed to activate it. Restart the system via the Server Monitor, then restart the Administration Client and load your saved CRS workflow. It should now show a **New VSV result** indicator that you can activate if you wish to use it in CRS rules.

4. To close the configuration editor window click the **Close** button. Then your new configuration item CR3DBHelper.ini is available for all Machines, Instance Types and Instance requesting this configuration item.



Factoring in external information

External information can be used in the same way. The chart below can help you determine where the best spot to store that information is, assuming it is not in the list of risk indicators already.

First, you will want to specify the type of information. Is it check related or account related? Account related information will be stored in the reference (SignBase) database since it will need to be reused for all items of the same account. Since the reference data will be historized, doing frequent changes on that part will quickly grow your database to an unacceptable size.

i Any changes to the reference data are stored in the system. Old data is kept. Items are never deleted, just marked as such. Thus, you can give the system a specific date in the past and it will be able to show you the information on the specified account as it was valid on that date.

It only makes sense to store external data in the reference tables if it doesn't change often. If it does, you will probably be better off to ask Kofax to create a custom database table and risk indicator for you.



Item (check) related information goes to the transient SignCheck database part. If it is available via an external API to other systems and it is not needed for further reference (reports, etc.), it doesn't need to be stored at all. Other information can be stored in the results table.

In the graph above the green destinations are risk indicators you can configure yourself (even though you might need to ask Kofax to write a custom loader program for them). The orange ones need a custom module.

I recommend you contact your Kofax sales representative and discuss your specific set up with him.

Wrap up

CRS is highly configurable. The CRS editor controls not only the rules used in routing and decision making but also the way FraudOne visual clients work.

You can define the processing queues to be used and the way visual clients display items while working on those queues. Queues can be defined by BNO.

CRS has a list of predefined risk indicators that can be enabled for rule use. This list can be extended if the customer specific installation requires it. Depending on the data needed risk indicators can either be configured by the FraudOne user, or Kofax can create a customer specific extension to access the data.

Chapter 10

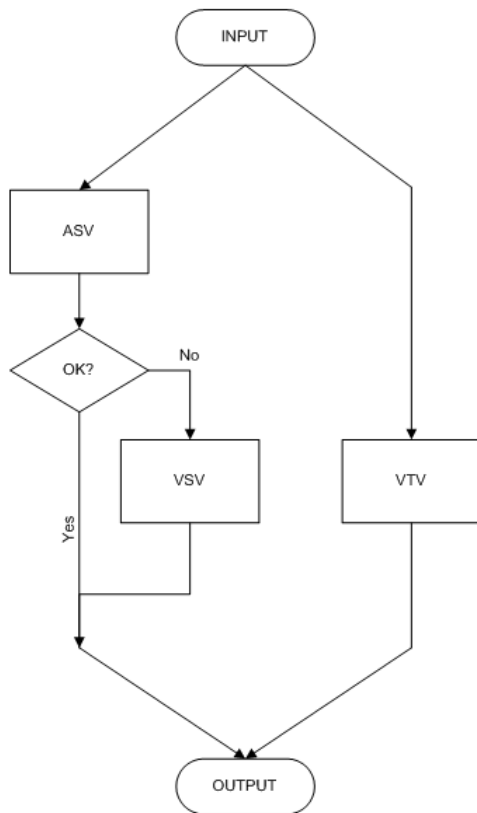
Doing it as a parallel process

What is the goal?

Did you wonder why you can specify more than one target inside one decision? The purpose of this chapter is to show you one of the reasons. We will design a workflow with queues that do parallel processing. This is sometimes desirable, due to the fact that throughput through visual queues cannot be well controlled. Also, sometimes users on some queues will work in different time zones than others. The drawback in using parallel processing is the added complexity in workflow design. You will want to avoid it where possible. Therefore, consider this chapter optional and follow it through if you really like a challenge.

Some banks want to have some or all of the checks technically verified. For the purpose of this tutorial, we will assume that all checks processed will have to be visually reviewed by a user (check the account information, funds available etc.). The group of users doing the technical verification will be different from the signature specialists doing the visual signature verification. They will get their own queue to work on, we will name it VTV (visual technical verification).

You could add the VTV queue before VSV having a simple workflow that moves items from INPUT to ASV, to VTV, then optionally to VSV and finally OUTPUT. That would be the easy way to do it. On the other hand, both VSV and VTV are visual queues where users process items. The number of users that process checks in each queue can vary, as does the throughput they provide. Users in VSV might have to wait for items from VTV to become available. Fortunately, FraudOne can also do parallel processing. This example will do that by processing ASV, VSV and VTV at the same time. The resulting workflow looks like this:



Signature and technical verification will be done in two parallel processing branches. Signature will do ASV and VTV, technical only VTV. Items coming from the INPUT queue will be placed into the ASV and VTV queue at the same time (think of it as splitting the item).

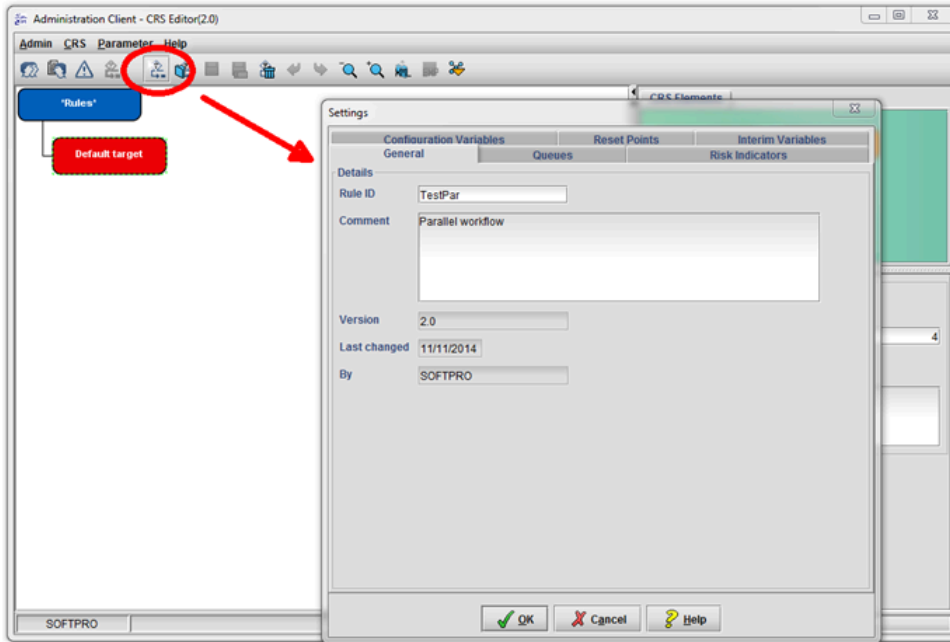
ASV and VTV now both have a copy of the item and can process it in parallel, at the same time. This makes the system a bit more efficient, since one engine/user does not need to wait for the other to complete. It is also a good way to merge in external results, where you can't control the arrival timing of the items.

The only thing you need to take care of, is merging the copies back together. Bear in mind: One specific item can only pass through a queue once. This is an important point, because it accounts for about 80% of the problems you will encounter when using parallel workflows. If you do not merge the copies back together, two copies referring to the same item will show up in the OUTPUT queue, causing an error. In the workflow above you will merge the halves after both ASV + VSV and VTV have completed on the item. From this point on, only one reference per item will continue its journey to the output queue (or any queues we might add later).

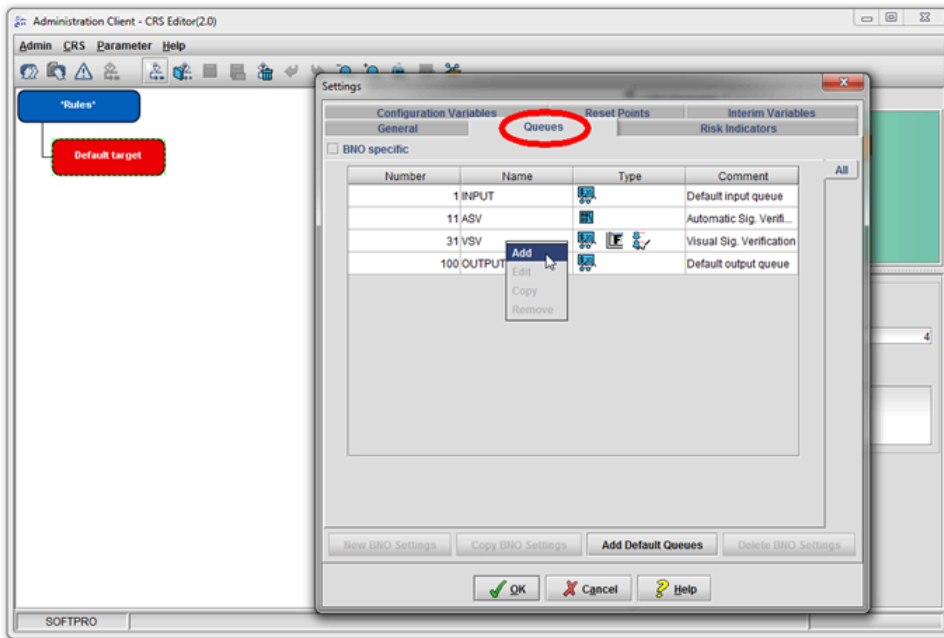
Another point before going on: It is much easier to write simple, non-parallel workflows. You will do it here to show you some features, but you might prefer to stick with the non-parallel variants until you are familiar with CRS and know how to find and correct errors.

Set up queues, indicators and other elements

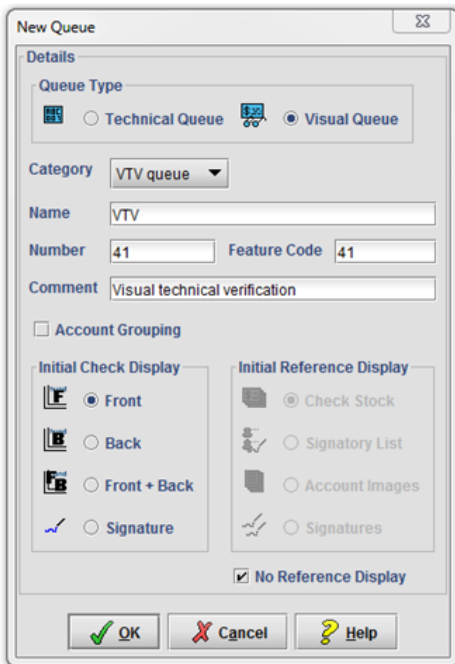
Start the Administration Client and move to the CRS editor to create a new workflow.



First, you will add the queues used. Create an ASV, VSV and VTV queue. Use the same settings for ASV and VSV as you used before. Create the VTV queue.

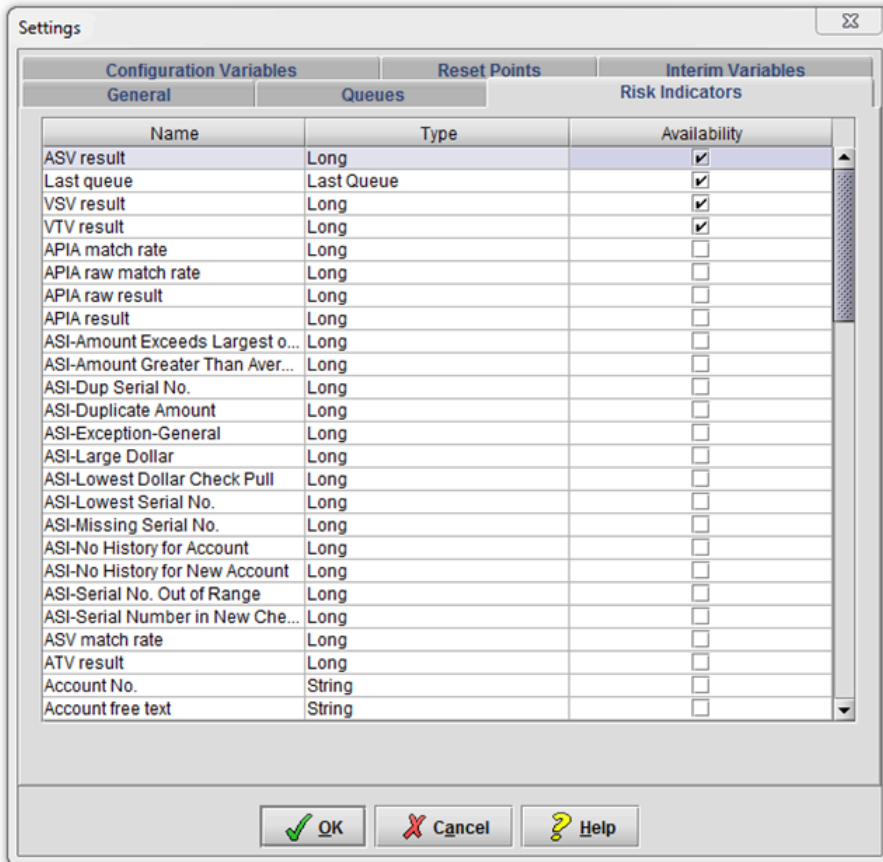


Right-click into the queue list and select **Add**. Add the VTV queue as shown below:



The display for the queue is set to show the front side of the check (not just the signature area, like VSV), without account information.

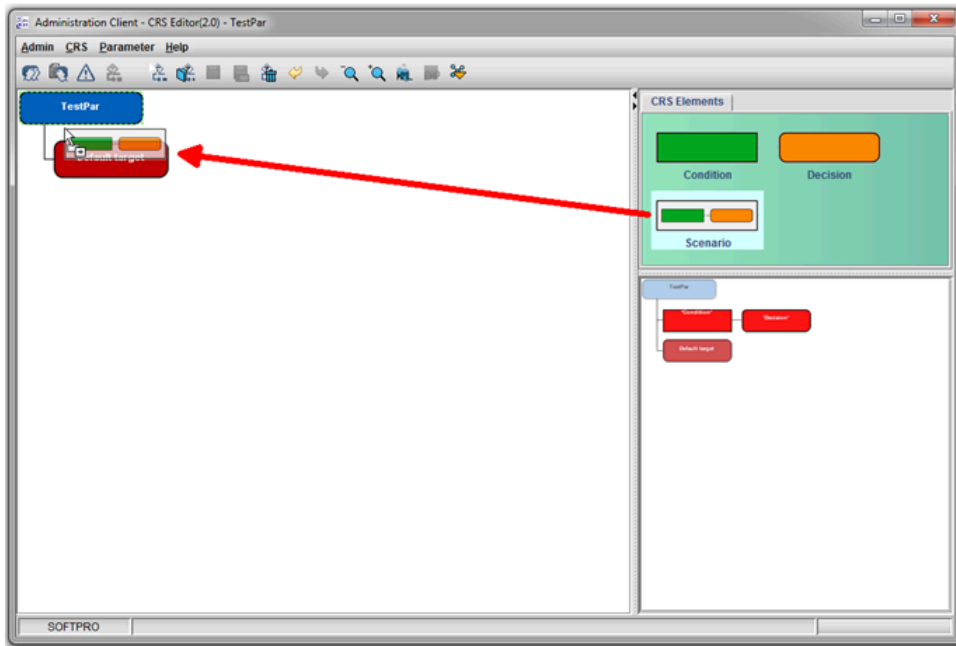
After you have finished setting up the queue, move to the **Risk indicators** tab. From the list of indicators, select **ASV result**, **Last queue**, **VSV result** and **VTV result**:



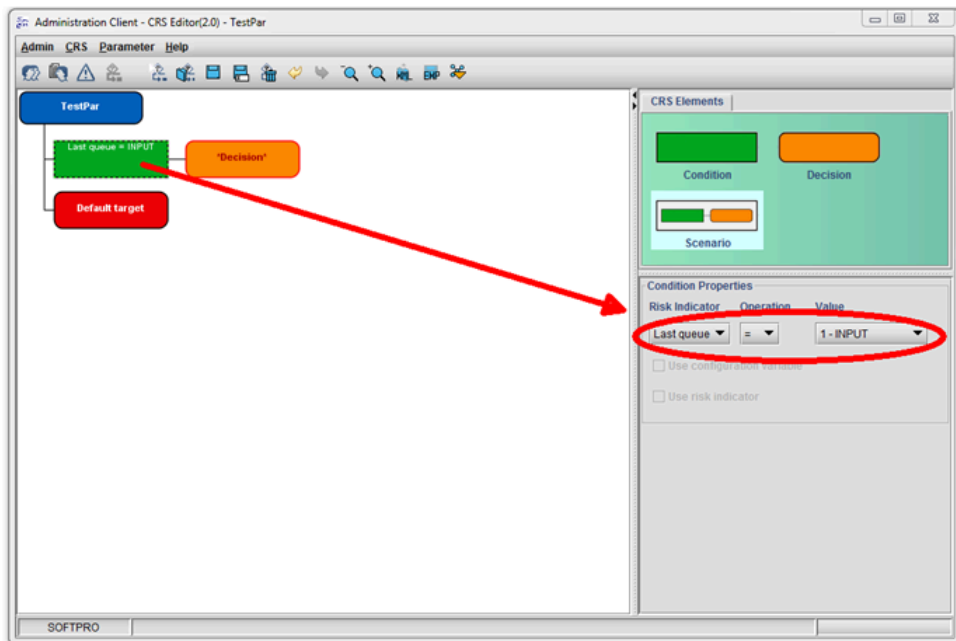
You can close the properties window and return to the rule graph.

Split items

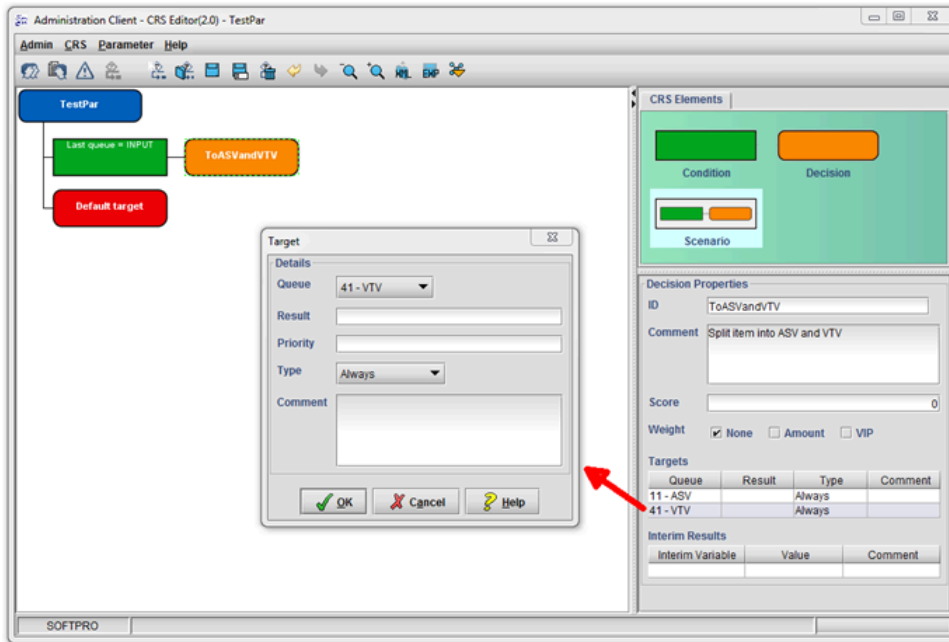
What you now need to do is, split items coming from INPUT to ASV and VTV at the same time. Create a new scenario and apply it to the input queue:



Select the empty condition and change it to **Last queue = INPUT**:



In the decision properties on the right side set the name to **ToASVandVTV**. Change the first target in the **Targets** list to **ASV** and the **Type** to **Always**. Now you must insert another reference of the item into the VTV queue too. Right-click on the target list on the bottom right and select **New target**. Create one for the queue **41 - VTV** of the type **Always**.



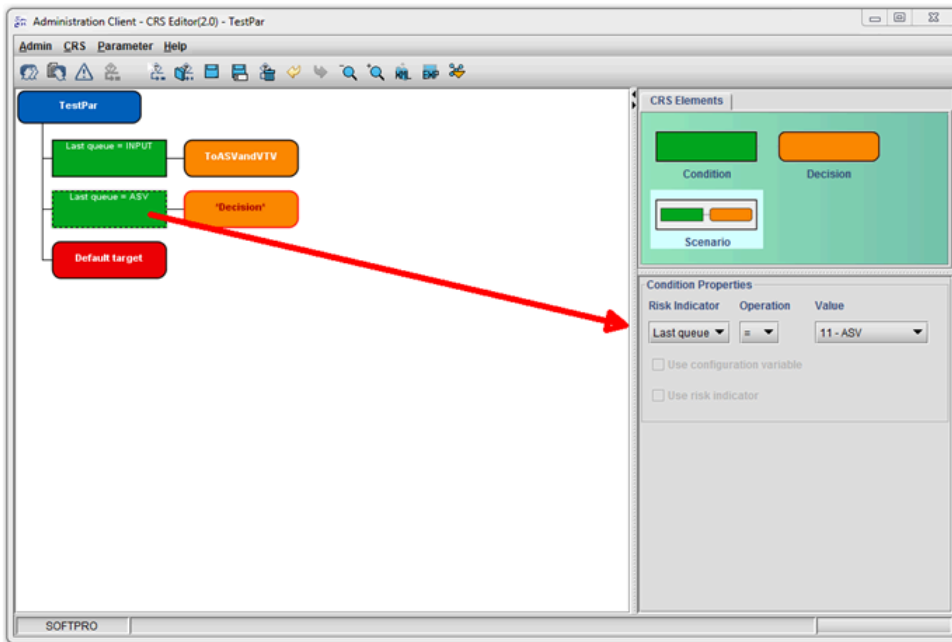
You now have two targets in the same decision that always apply: one leading to ASV and another to VTV. Items coming from INPUT will be put both in ASV and VTV at the same time.

That was the easy part. The trick is to merge the parts together again.

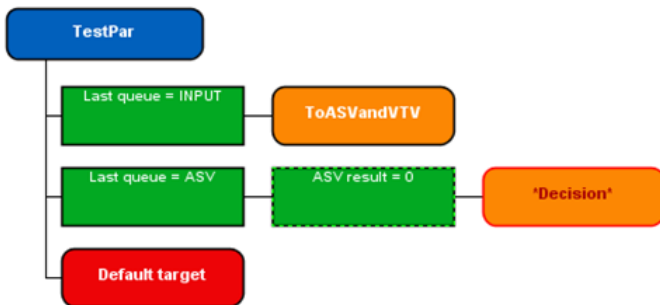
Route items

The routing from ASV to VSV is standard. We will do it the same way as in the previous example. If the ASV result is good the item can be accepted, otherwise the item will be routed to VSV for a visual review.

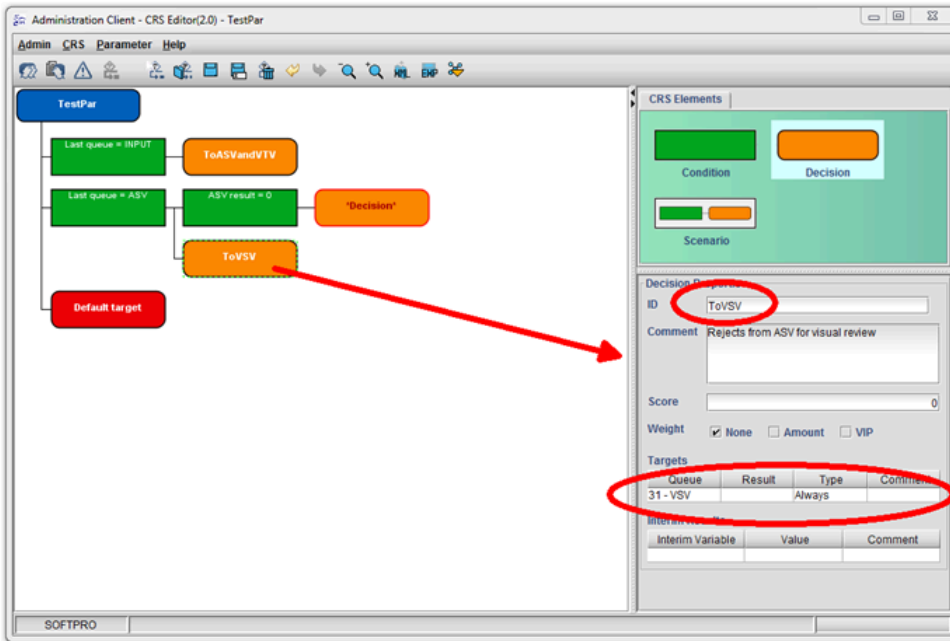
Add another scenario and set the first condition to read **Last queue = ASV**. This scenario now applies to the items coming from the ASV queue:



Add a condition reading **ASV result = 0** to check for a good signature verification result:



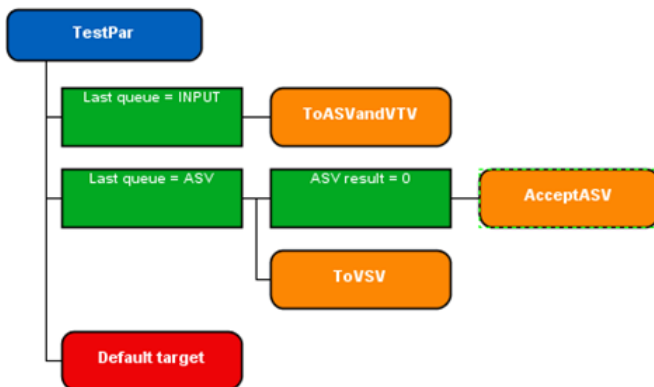
Add another decision below **ASV result = 0**. Name it **ToVSV** since it deals with rejects from ASV that will have to be reviewed in VSV. Change the target to point to the VSV queue, type **Always**:



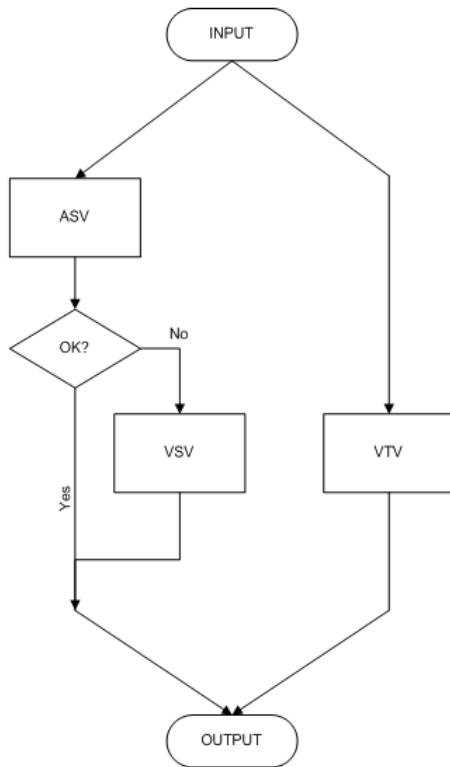
Since this routing is completely inside one of the parallel branches, there is nothing special to consider. You can handle it normally.

Merge items back together

Now consider the decision at the end of the scenario, behind **ASV result = 0**. You can name it **AcceptASV**.



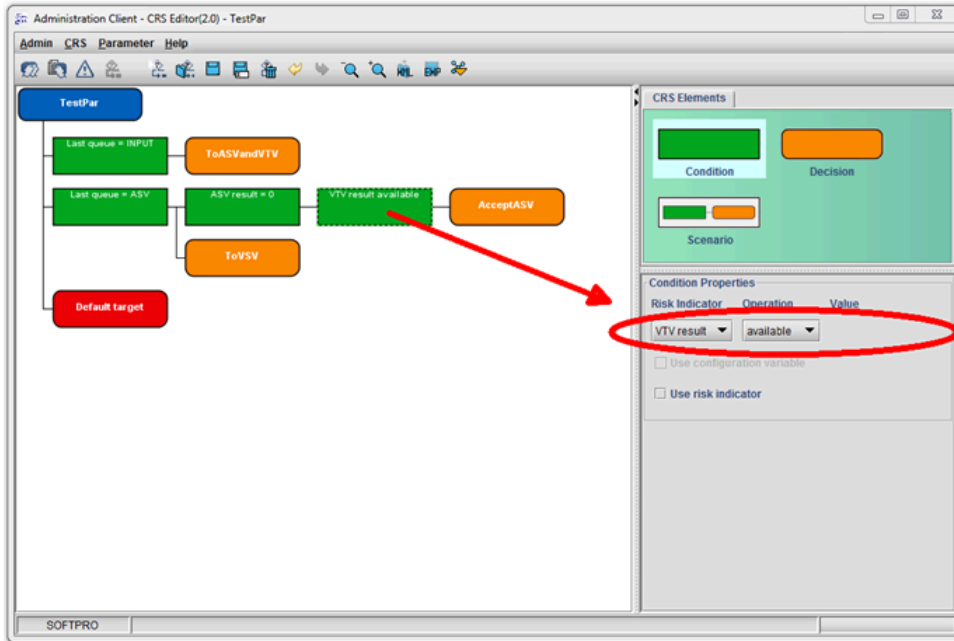
In a non-parallel workflow items with a good ASV result would be routed to the output queue and be accepted. In our parallel workflow, you have to first merge it back with the part routed via VTV before you can continue the routing.



Remember that an item can pass a queue only once. You can't put two references of the same item in the output queue.

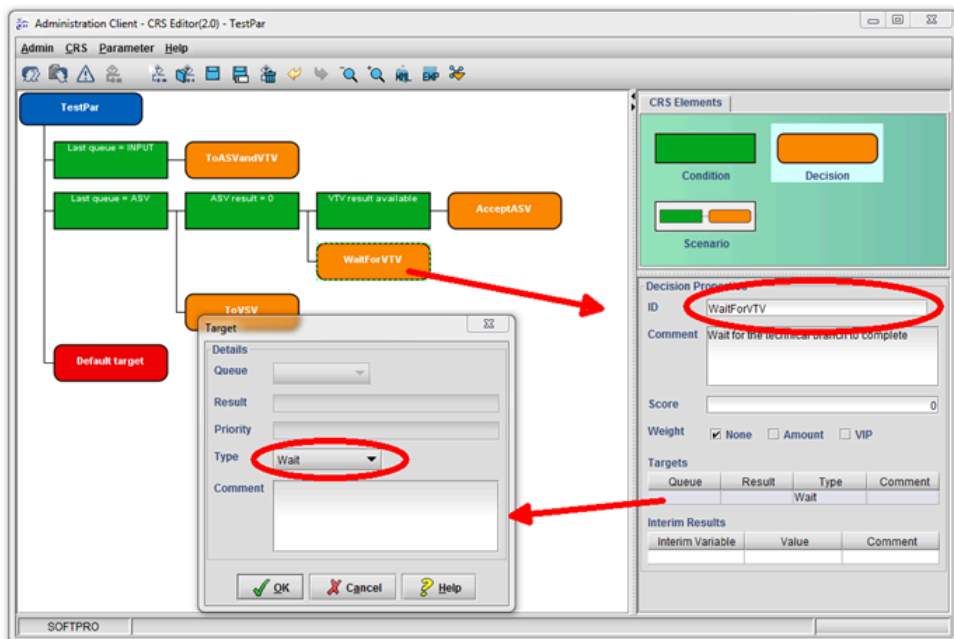
What do you need to do then? First, you have to determine if the copy in VTV has completed processing. If not, you will have to wait. If it did, you can merge the item and route it to OUTPUT with a final result. The final result will depend on both the signature and technical verification results. If both results are good, the item can be accepted, if any of them fails, the item will be rejected.

How can you determine if the technical branch has completed? Since it consists only of the VTV queue, you can check for the availability of the VTV result. If the result is available, the item has completed the branch. Let us add this check. Insert a condition before the **AcceptASV** decision and change it to read **VTV result available**.



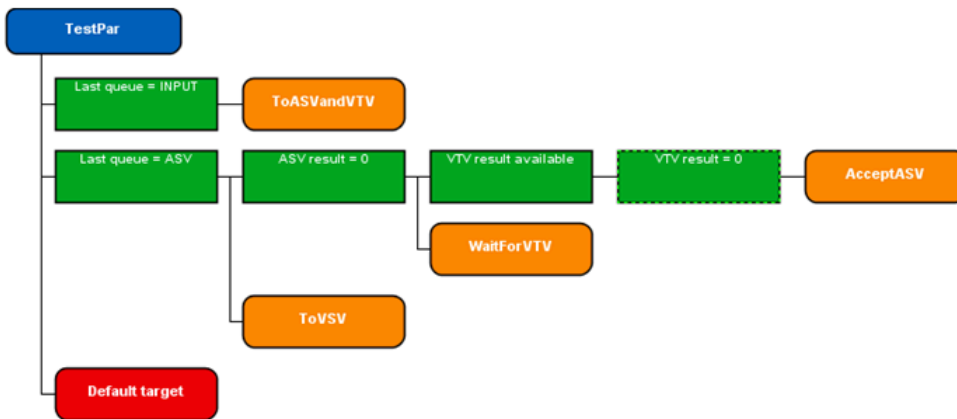
This condition will now only be passed if the VTV verification has completed. If yes, we can reach the decision at the end of the scenario. What do you have to do if not?

If the parallel branch has not yet completed, you have to wait for it to do so. Let us add an according decision. Add a decision below **VTV result available** that will only be reached if not. Change the name to **WaitForVTV**. Open the target and change the type of the target to **Wait**:

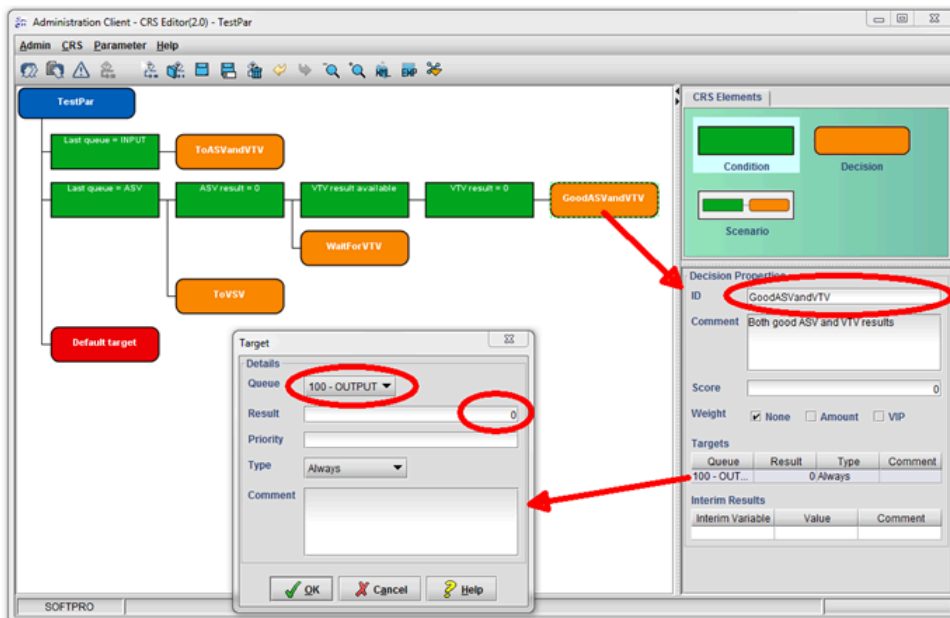


Now the item will wait indefinitely. Basically this copy of the item is not active any more. You have to rely on the other side of the parallel workflow, which will complete at a later time, to continue processing the item.

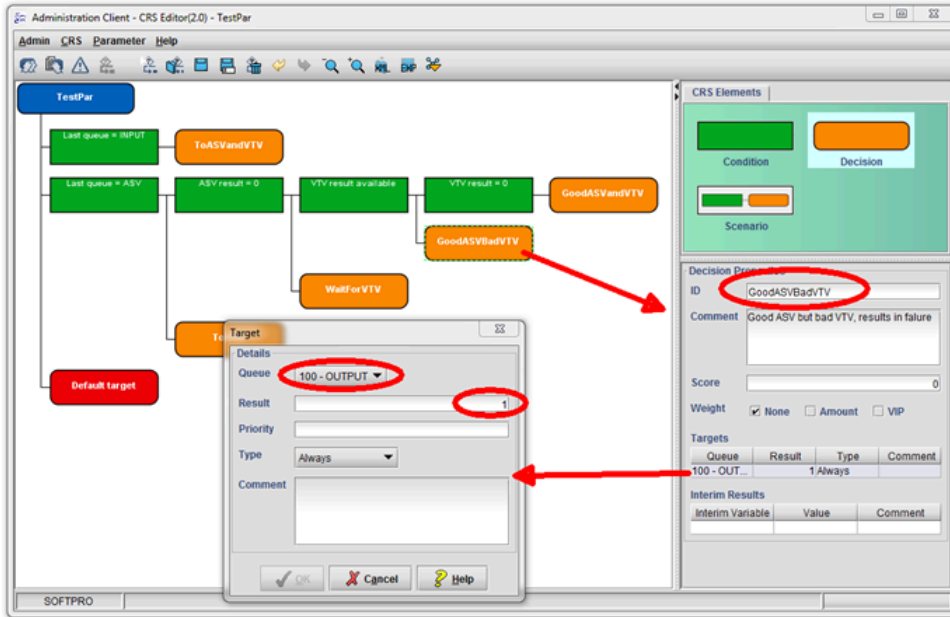
Let us return to the case where the parallel branch has completed and the VTV result is available. The **AcceptASV** decision could now process it further. If you look at the diagram above you will notice that the item will be routed to the output queue. But what will the final result of the item (which we always set for the output queue) be? The signature verification was OK, since we are at the end of the **ASV result = 0** scenario. But what is the VTV result? We will have to check. Add another condition before the **AcceptASV** decision and change it to read **VTV result = 0**:



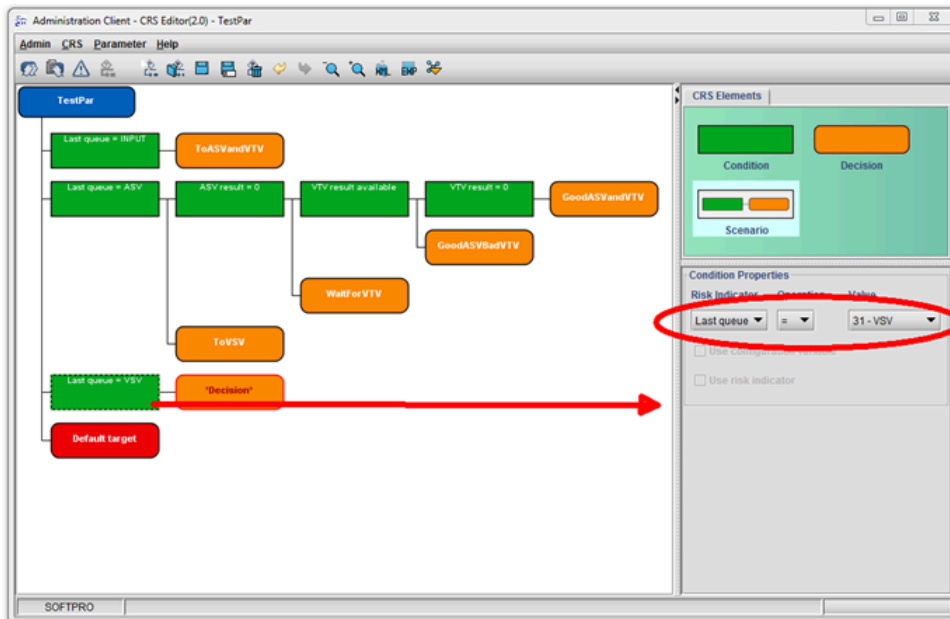
The **AcceptASV** decision can now only be reached if both ASV and VTV results are good. We can change the name to **GoodASVandVTV**, set the target to **OUTPUT** and the result to 0. This item is done:



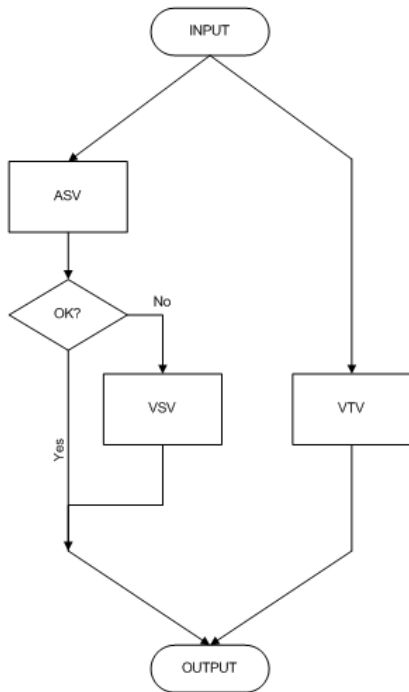
For the case that the VTV result is not 0 we need another decision below the condition to handle it. Let us name it **GoodASVBadVTV**. It will route the items to **OUTPUT** too, but with a result of 1 (fail):



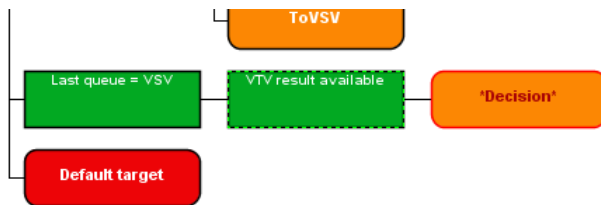
Seems we are done with the rules for the ASV queue. Let us proceed to the VSV queue. Add another scenario to the end of our rule graph and change the first condition to read **Last queue = VSV**.



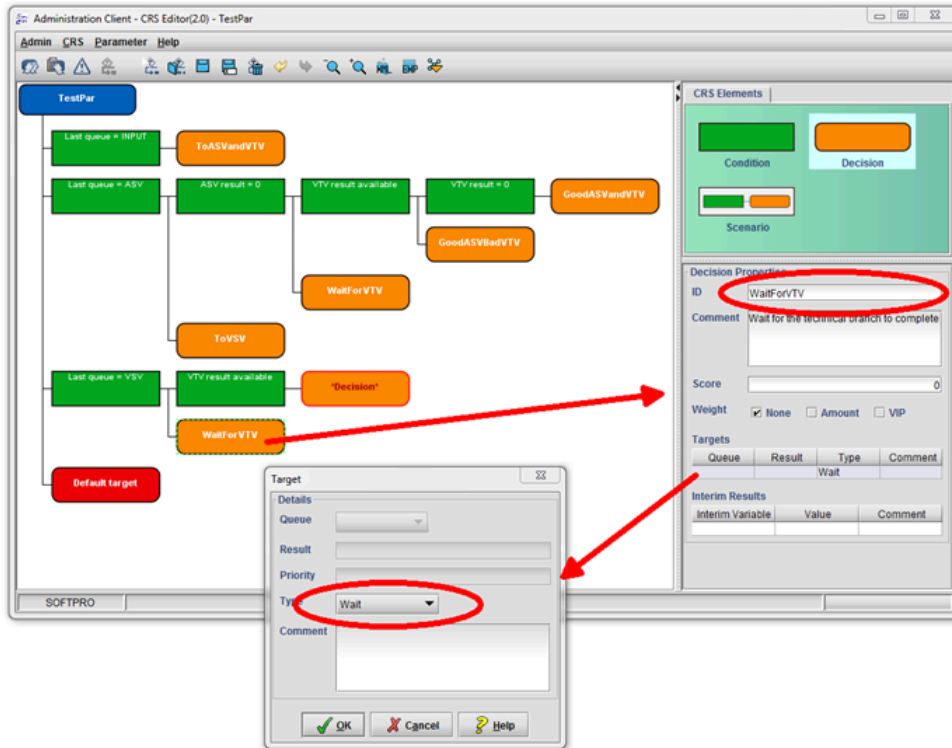
Let us review the graph again:



Regardless of the VSV result the processing in this side of the branch has ended and we will have to merge the item back again. To do so we have to check if the VTV branch has also completed. We will add another condition reading **VTV result available**:

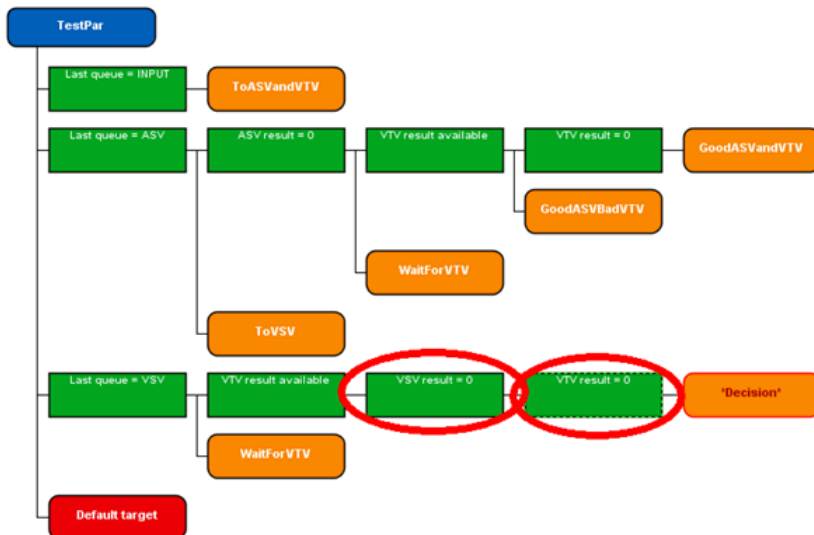


If the VTV branch has completed (and the VTV result is thus available) you can process the item. If not, you have to wait. Add a decision below **VTV result available**, name it **WaitForVTV** and set the target to **Wait**:

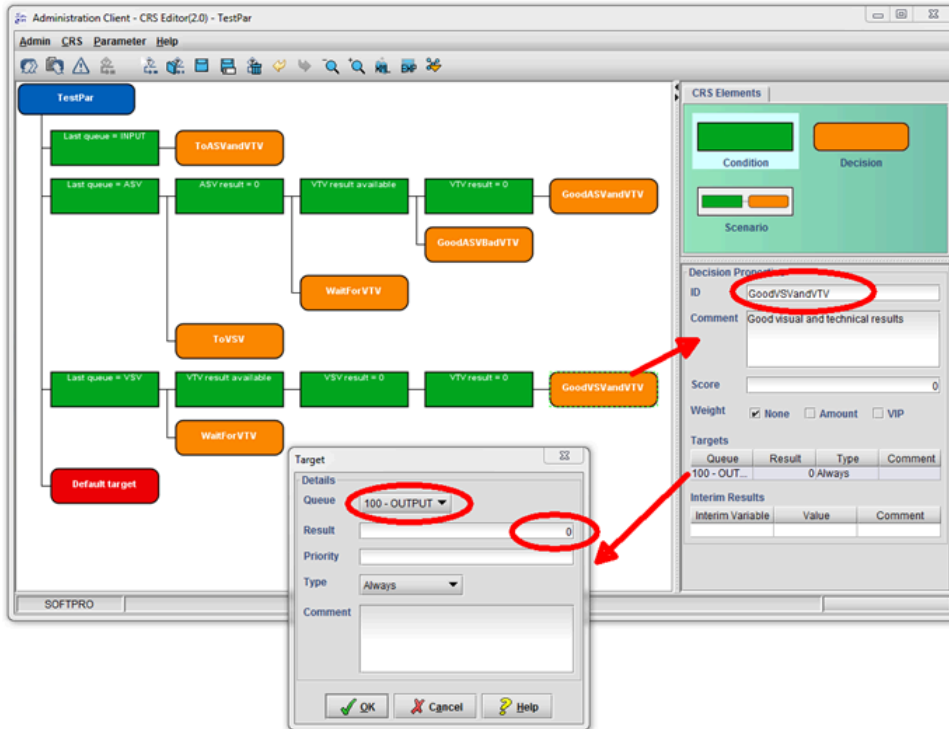


If the technical branch has completed, we can safely route the item to the output queue setting the final result. In order to do so, we have to check if both VSV and VTV are OK so the item can be accepted or rejected if any of the two has failed.

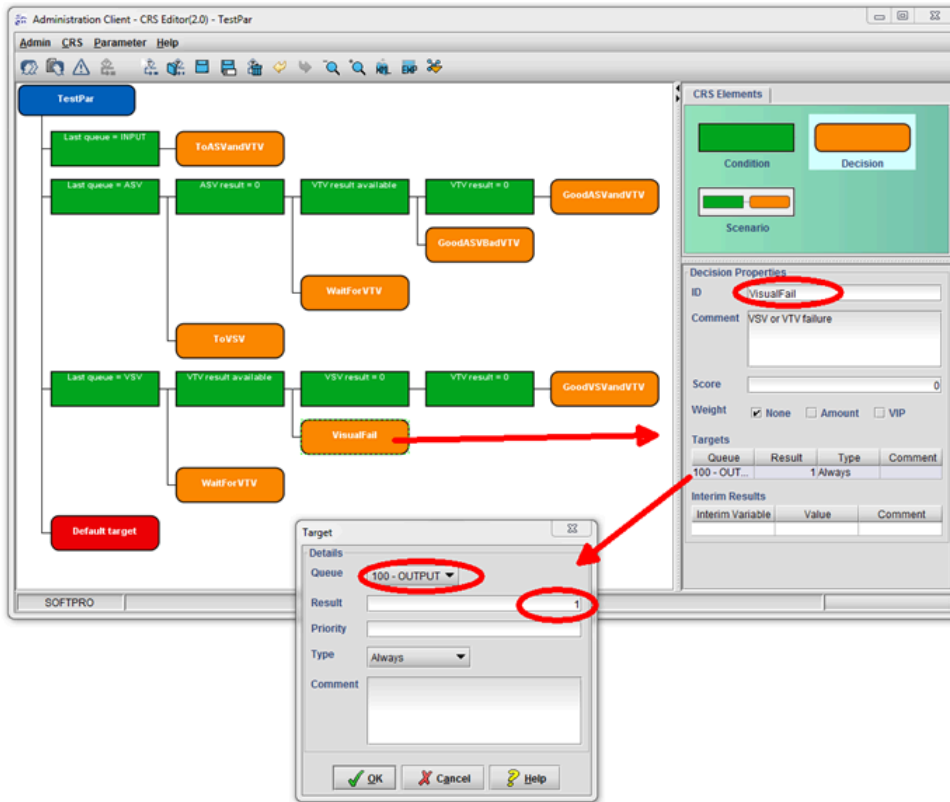
Add two conditions reading **VSV result = 0** and **VTV result = 0** to the scenario:



The decision at the end can now only be reached if the item has passed both VSV and VTV. We will name it **GoodVSVandVTV**, set the target to **OUTPUT** and the result to 0:



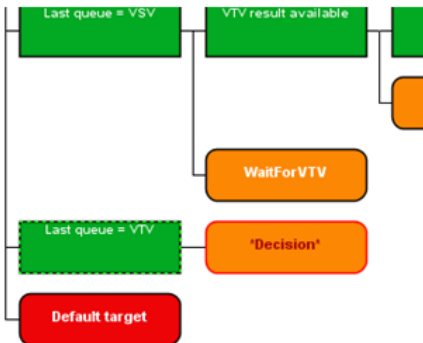
To cover the case where any of VSV and VTV failed and the item has to be rejected in **OUTPUT** you must add a decision below **VSV result = 0**. Change the name to **VisualFail**, set the target to **OUTPUT** and the result to 1:



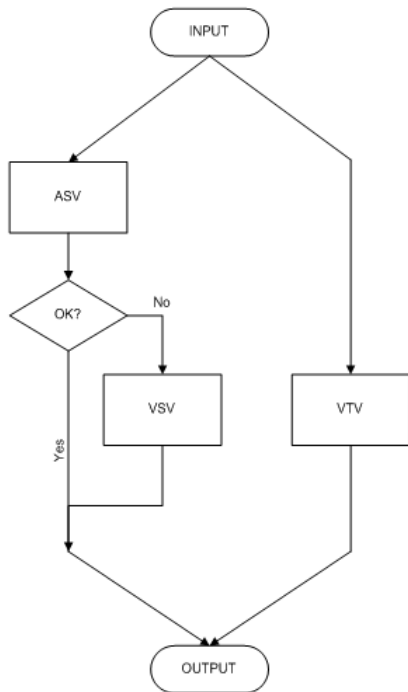
This also completes the processing for the VSV queue.

Use interim variables to merge parallel branches

The last queue left that you must define is the VTV queue. Add another scenario at the end of the rule graph and change the first condition to read **Last queue = VTV**:



Since this is the only queue in the technical branch you will have to merge the item with the visual branch, regardless of the VTV decision. Now here comes the tricky question: How do you determine if the visual branch has completed?

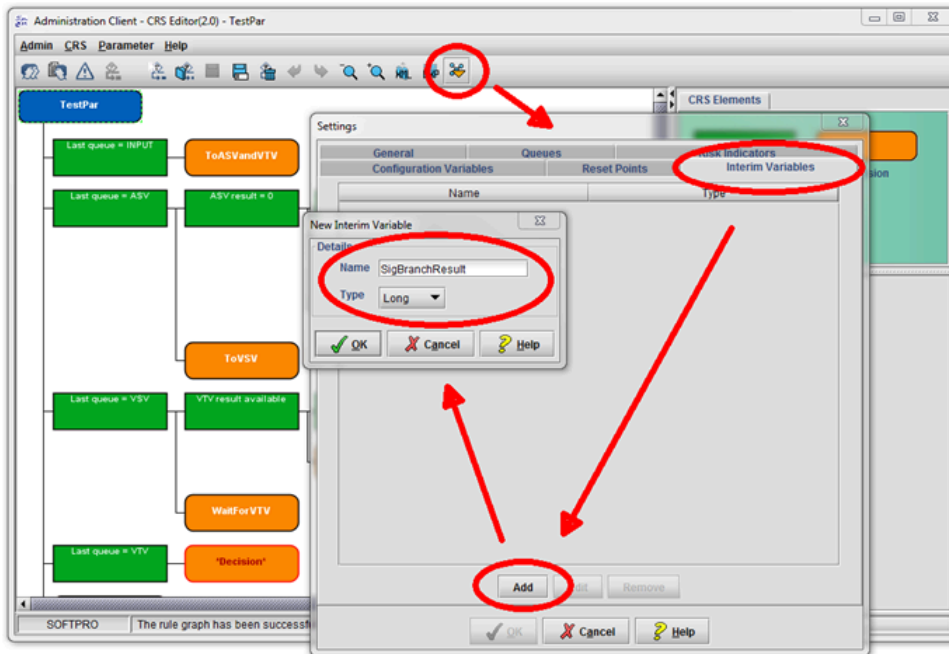


Unfortunately, the visual branch does not consist of only one queue, and not all items pass through VSV. There is no single result where you can determine the availability and deduct that the signature branch is done. You would have to replicate all conditions that process the items in the branch to determine completion (ASV result is available, is it ok, if not is the VSV result available, etc.).

Fortunately, there is an easier way to handle this case. You can use interim variables. An interim variable is essentially a risk indicator that you can define in CRS and set inside any decision. It can be queried and used like any other risk indicator.

For our purposes we will define an interim variable named **SigBranchResult**. We will set it to the result of the signature branch and check the availability to determine if the signature branch has completed.

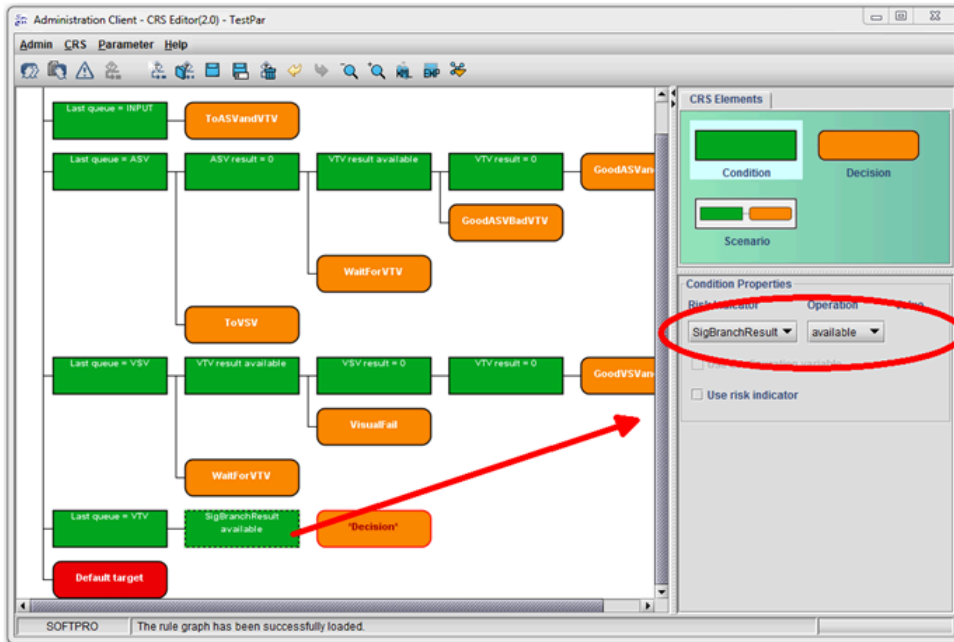
Open the rule graph properties by clicking on the appropriate icon. Select the **Interim Variables** tab and add a new interim variable by clicking the **Add** button:



Name the variable **SigBranchResult** and set the type to **Long**. We will use **Long** instead of **Boolean** since we don't want to just store the fact that the signature branch completed, but also the combined result of the branch (ASV and/or VSV).

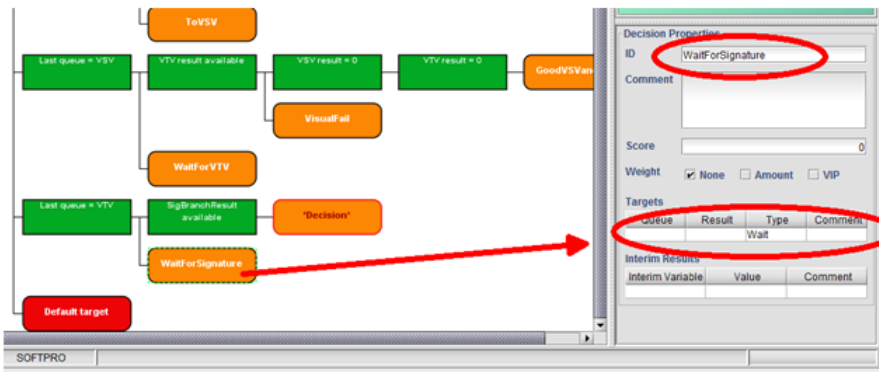
Close the properties window by clicking **OK** and return to the rules graph.

Now that you have the **SigBranchResult** variable it is easy to check it. Add a condition after **Last queue = VTV** and set it to read **SigBranchResult available**:



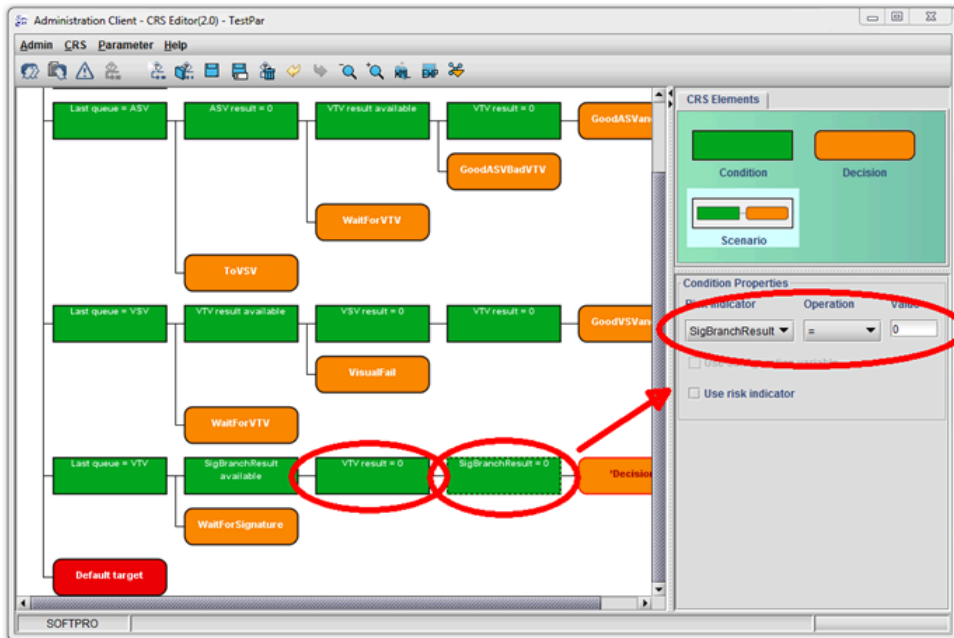
If **SigBranchResult** has been set the signature branch has completed and **SigBranchResult** will contain the result of the signature verification. The item can be routed to the output queue.

If the **SigBranchResult** has not been set yet, the signature branch has not been completed and the item has to wait. Add a decision below **SigBranchResult available** and set the target type to **Wait**:

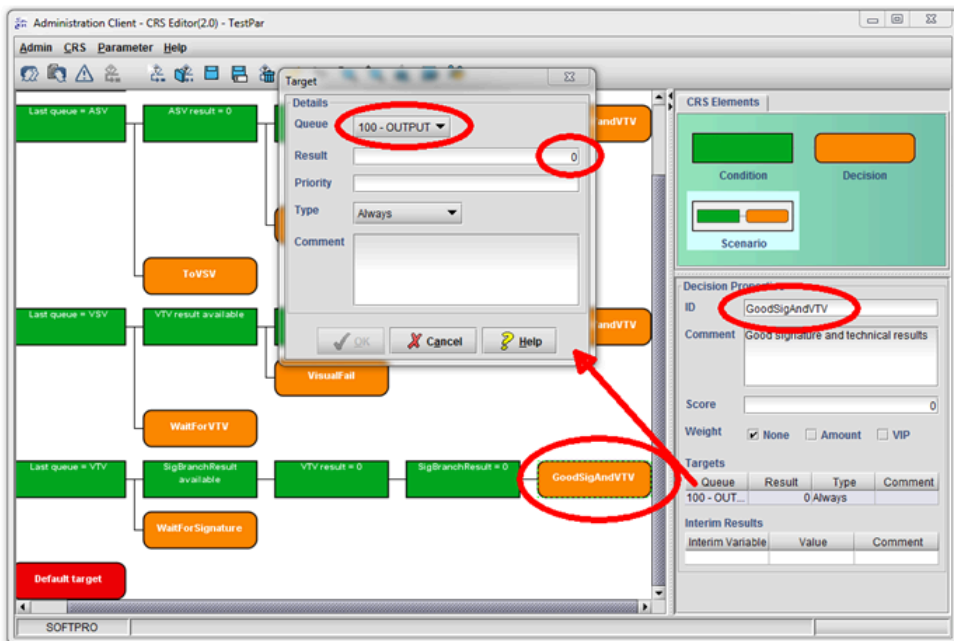


You can use **SigBranchResult** and the VTV result to determine the final result for **OUTPUT**.

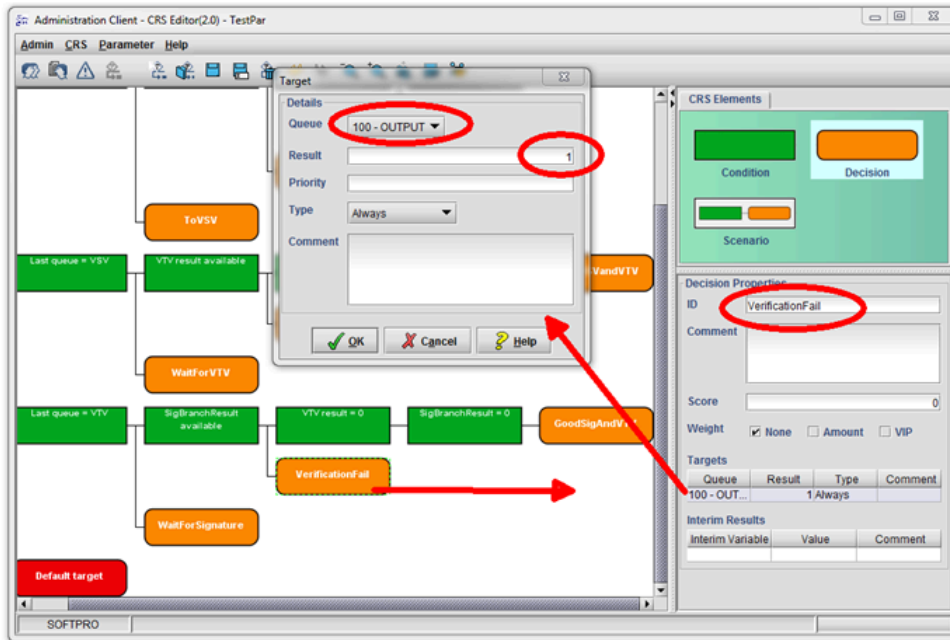
Create two new conditions after **SigBranchResult available** and change them to **VTV result = 0** and **SigBranchResult = 0**:



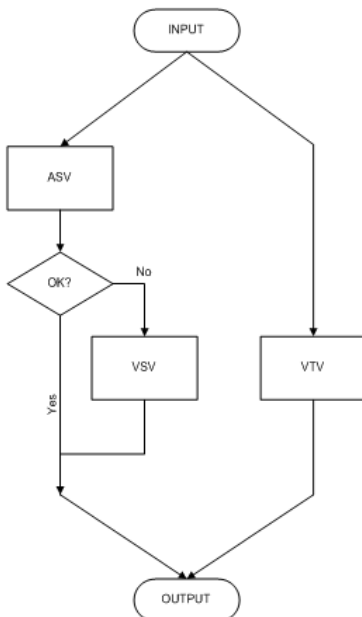
The decision at the end of the scenario can now only be reached if both signature and technical results are OK. You can route these items to **OUTPUT** with an **accept** result of 0:



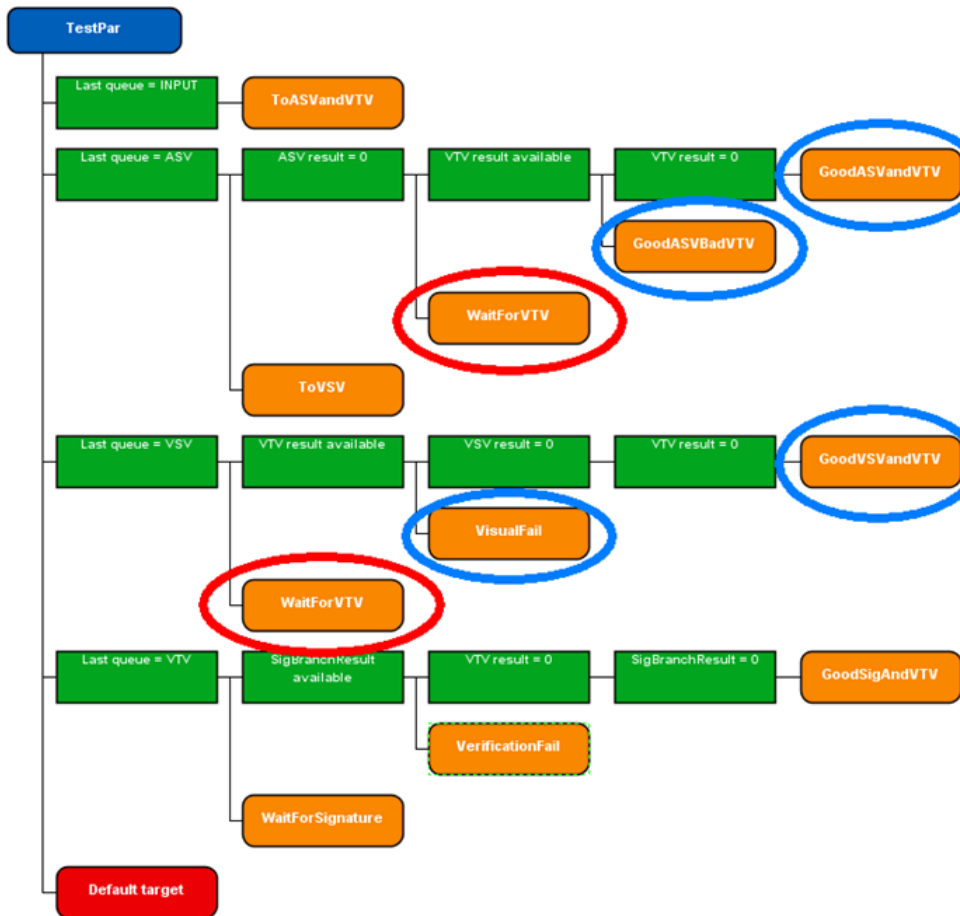
If any of signature and technical results are bad, you must route the item to **OUTPUT** with a failed result. Create a new decision below **VTV result = 0** and name it **VerificationFail**. Route the items to **OUTPUT** with a result of 1:



If any of signature and technical results are bad, you must route the item to **OUTPUT** with a failed result. Create a new decision below **VTV result = 0** and name it **VerificationFail**. Route the items to **OUTPUT** with a result of 1:

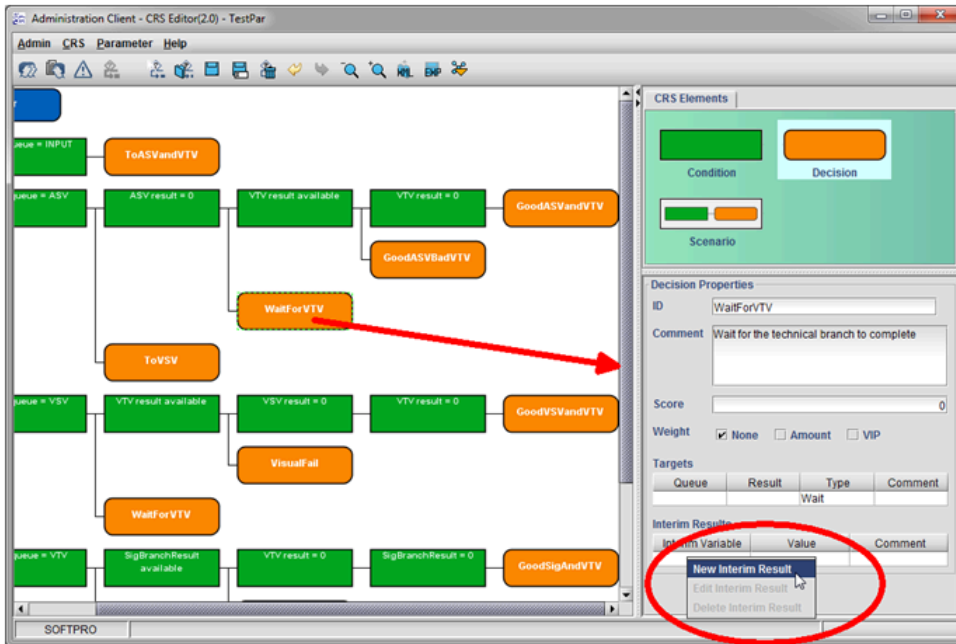


The signature branch can complete if ASV has a good result, or after VSV, regardless of the VSV result.

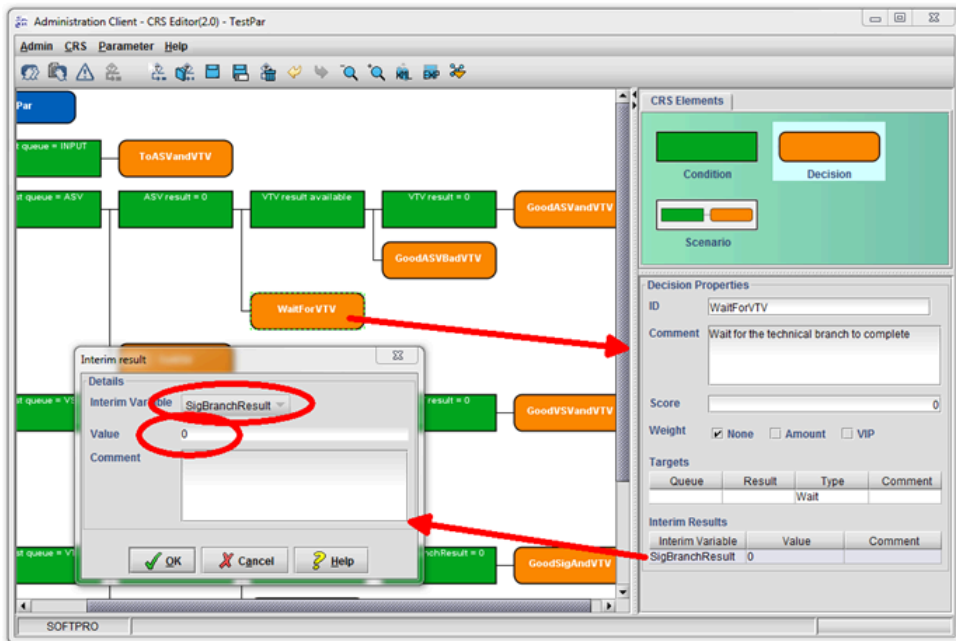


All decisions where the signature branch completes have been marked above. But do you need to set the **SigBranchResult** interim variable in every one of those? You could if you also want to use that variable for documentation. But only some of those decisions deal with the case where the signature branch completes before the technical branch. In all decisions marked blue, the technical branch has already completed and the item can be routed to **OUTPUT**. Only the decisions circled red deal with cases where the signature branch completes but the technical one isn't ready yet. Here you have to set the **SigBranchResult** variable, which will be used by the technical branch when ready. As a rule of thumb, you have to set the branch result whenever you let the item wait (setting the target to type 'Wait').

Select the first **WaitForVTV** decision and right-click on the **Interim Result** list. Select **New Interim Result** from the menu:



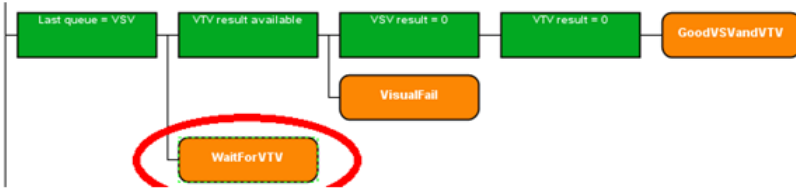
Select the **SigBranchResult** variable and give it a value of 0:



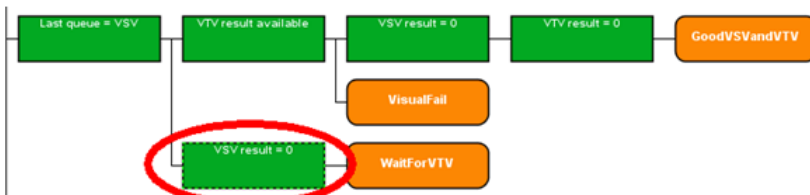
We use 0, since 0 indicates a good result and at this part of the rule graph we checked that ASV had a good result.

If you now look at the second **WaitForVTV** decision you will notice that we didn't bother to differentiate between the possible VSV results at that time. After all, we had to wait, regardless of

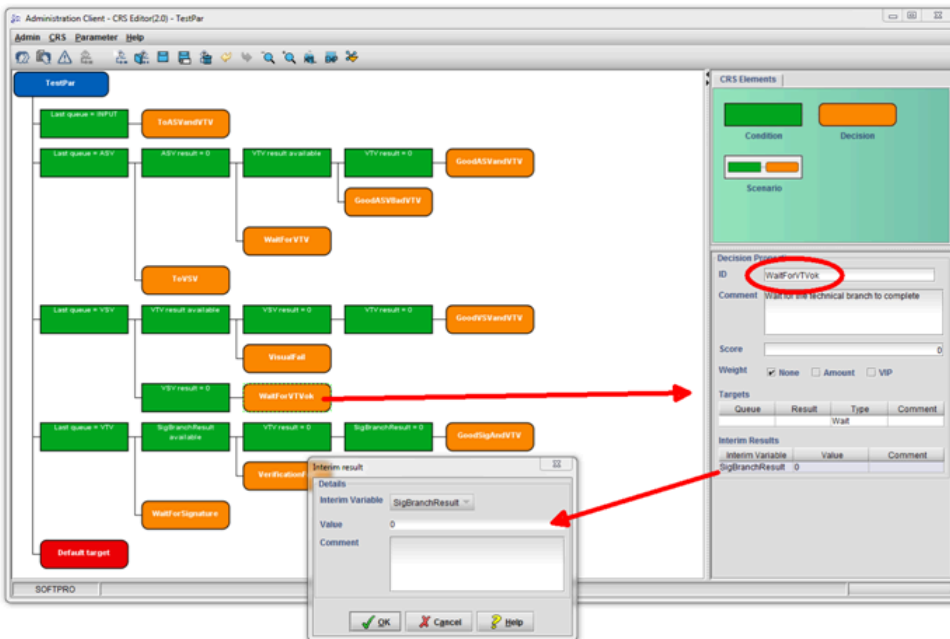
the VSV result. But if we want to set a correct **SigBranchResult** we need to know the outcome of the VSV verification.



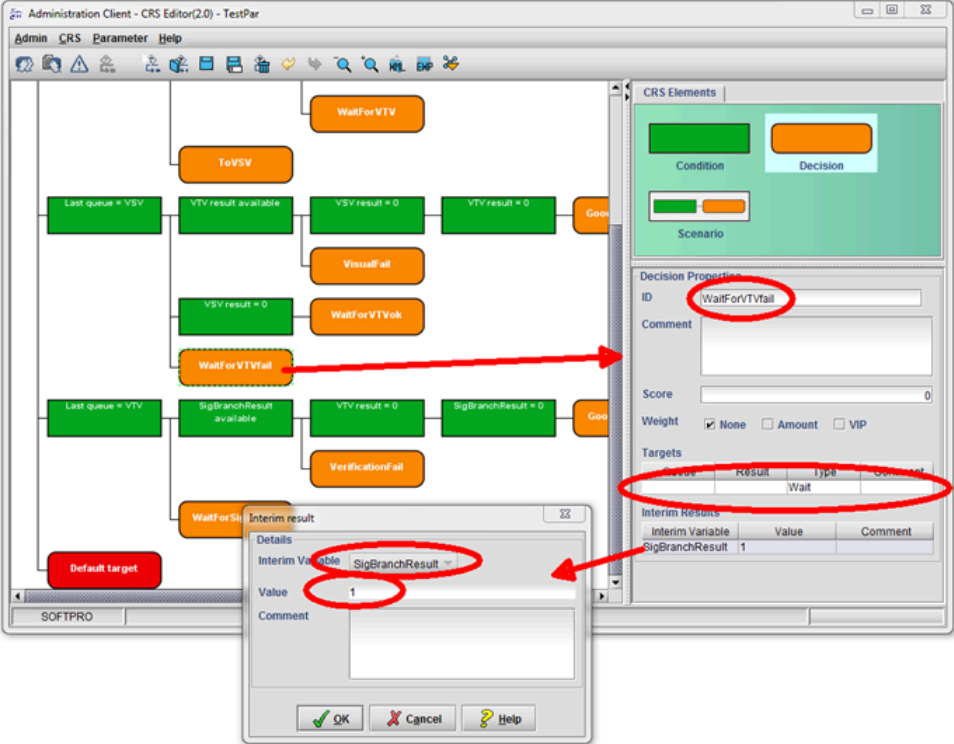
Add another condition before **WaitForVTV** and change it to **VSV result = 0**:



This condition now makes the **WaitForVTV** condition at the end only reachable if the VSV result is OK. Change the name of that decision to **WaitForVTVok** and set the interim variable **SigBranchResult** to 0:

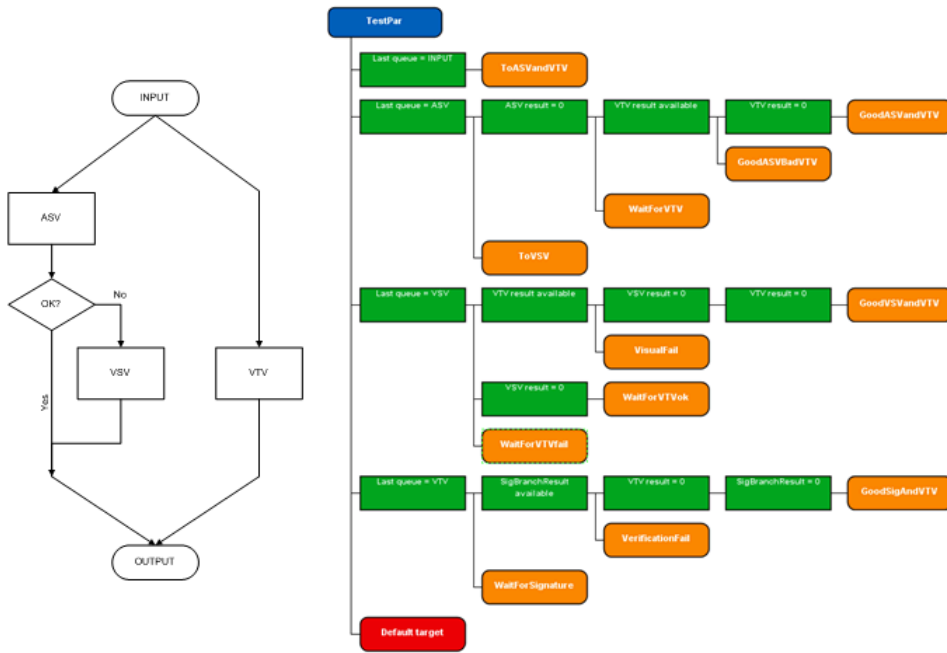


You now have to add another decision for the case that VSV verification failed below the **VSV result = 0** condition. Name it **WaitForVTVfail**, change the target type to 'Wait' and set the **SigBranchResult** to 1:



You have now set the **SigBranchResult** for all relevant cases. You can save the rule graph, activate it, restart the system and give it a spin.

Wrapping it up



To recap what happens with one item:

- New items are placed in INPUT. The first CRS rule applies and the **ToASVandVTV** decision is carried out. One copy of the item is placed in ASV, another in VTV.
- The ASV engine verifies the signature and either completes the signature branch or forwards the item to VSV.
- Both VSV and VTV users get to work on the item at the same time.
- The first item copy that completes one of the branches is left waiting (no action is taken on the item) by one of the decisions **WaitForVTV(xxx)** or **WaitForSignature**.
- In any of the 'wait' type decisions in the signature branch, the interim result **SigBranchResult** is set. The conditions in the technical branch can use this variable to determine the status of the signature branch.
- Only after the other branch completes will the processing continue. The **SigBranchResult** variable is used to query the state of the complete signature branch. A similar variable could be used for the technical branch, but is not needed since the branch only has one queue.
- Only one copy of the item will continue after the branches has been merged, in this case to the output queue.

Wrap up

You can use CRS to design parallel workflows. This is useful if the timing in the parallel branches differs.

There is no limit as to how many parallel branches you can use. You can split an item in two, three, four hundred, etc.

If you split an item, you need to merge it back together. Do not forget an isolated item copy. An item is only allowed to pass a queue only once. Putting the same item (or another copy of it) a second time into the same queue will lead to a system error.

You split an item by adding multiple targets to a decision.

You merge item copies by using the **Wait** target to drop one copy and continue processing with another.

Interim results help in recording and querying the state of a complete branch.

Chapter 11

Best practice

Gather information

The problem with rule editing is not how to use CRS to write rules, but rather the content of the rules themselves. Therefore, before you begin, you should consider carefully preparing the information you need for rule design.

This information usually falls into two categories:

1. How to identify fraud.
2. How to deal with fraudulent items.

To identify fraud, you can either rely on your past experience (or that of your fraud analysts), or gather statistical data that can help you narrow down your search. Consider the available list of risk indicators, plus the ones you like to add on your own and get statistical data on checks to try to identify relevant indicators. It is important that the data should be statistically relevant (say not only a Monday worth of checks, or only checks from only one BNO). Also the data needs to plot the ratio of fraud against good checks. Looking only at the fraudulent items is not enough. Your fraud analysts and, if needed, Kofax can help you gather the data.

i If you only look at fraud, you might find that, for instance, most of the fraud items are in the \$100 to \$1000 range. That doesn't tell us much, because it is likely that the range will also contain the biggest number of good checks. The fraud vs. good item ratio is important.

It might help running checks through the system for a while to collect information on ASV and APIA results and rates, PAD return codes, etc.

After you identified relevant risk indicators and their range you can start designing rules to catch and score those items.

Now that you know how to identify fraud, switch to considering what to do with it: Do you want to implement a visual workflow using verification clients, or do you just send the items to some other existing review system? How many queues do you need – do you want only one review or do you need a second review for high amount items? Do you have different groups of analysts working different items and therefore need different queues? How do you name them?

With all that information available, you can start by defining queues, risk indicators and the rules themselves, as described in this tutorial. Also consider an iterative refinement stage where you compare results and refine the rules until you obtain the desired effect.

Here is a check list helping you in the information gathering stage:

- Information on system throughput:
Number of items to be processed, time available, distribution of the items throughout that time.
What is the processing capacity of your fraud analysts?
- Information on fraud identification:
What risk indicators are needed, what is the fraud vs. good items distribution for those indicator ranges, what other information helps you identify fraud?
- What FraudOne and external engines (ASV, PAD, APIA, ASI, etc.) do you need to produce the desired risk indicators?
- Are other banking systems interfacing with FraudOne?
Do you need to design loaders and/or queues for those systems?
- Do you need a BNO dependent set up?
Do you have multiple organizations operating independently, each using its own fraud processing?
- How does the fraud processing look for each BNO?
Do you have independent groups of fraud analysts that work a subset of the items? Do you need second reviews? How many queues will you be using?
- What is the output you need from the FraudOne system (results, scores, risk, other information)?
How will systems following FraudOne use the results?


Planning for errors

The CRS rules editor is a rather complex tool. It is designed to be flexible, which, unfortunately, also provides for many different ways to create errors. On the good side, CRS also provides some design concepts you can use to avoid or catch errors.

The most important one is that CRS does not allow items to get lost if you don't expressly order it to do so. Even a newly created rule set contains a default target. If an item parses through CRS and no rule applies, the system will use the default target on the item. It will also log that in the system log. Therefore, try avoiding using the default target as a regular routing object, but reserving it for error cases.

The default target is the last thing that can save a badly configured item and therefore should be the first thing you configure. The most important paradigm here is 'keep it safe'. If the default target does not work, nothing can save an item that gets to it because of a rule error.

In non-visual workflows (engines only) the best way to set the default target is to have it move the items to the OUTPUT queue and give them a result that signifies 'not processed'.

 The standard FraudOne output result for 'not processed' is 4.

Depending on the flexibility of the system behind FraudOne, you can also decide to change the result to 'accept' or 'reject' but using a unique error result is a better idea, because you can scan the putter file for it and check on the items that generated errors.

In visual workflows (the ones that use interactive visual clients), you have the additional option to catch the error items and try to solve the condition. You can set the default target to move the items

into a visual queue, where an operator can review and decide them, thus saving the checks in error. This is also an easy way of alerting the operator to the routing problem. There is, however, also a risk associated to this approach. Assume the error treatment queue is named **A**. If the routing error occurs while routing items out of queue **A**, the default target will happily place the item back into the queue **A** trying to correct the error. What looks like an infinite loop is actually not one, because the system will not be able to place the item back into queue **A** since the same item can't pass the same queue twice. That item is lost.

The important thing, therefore, is to pay special attention to any routing and rules you write as part of the error correction branch. If you decide to manually correct items using a visual queue, make sure the rules that process that queue are correct. Preferably move the items from the error correcting queue directly to OUTPUT, with as little rules as possible. Write those rules first and test them before you begin writing a complex rule set. If they work well, you will have a safety net in place that catches mistakes that might have been made in other parts of the rule set.

Rule ordering

While deciding an item, CRS parses the rules starting at the top and trying to reach the first decision possible. If all conditions in a branch are true, the decision at the end of that branch is reached and parsing stops. Any rules below will have no influence on the item whatsoever.

Therefore, the order the rules are written is important.

Consider this example:



An item with both bad ASV and APIA results will never reach the second rule because the first rule is also true (both results bad means also one of them is bad). If you use a construct like this assign items a different score (both bad has a higher score then only one bad), be very careful about the order you place the rules into the rule set.

There are some points that can help you with this:

- You can start rules with **Last queue = XXX** type of conditions. Branches of this rule will only apply to items coming out of that queue. If rules start with a condition like that, the order of these is not important because an item can't come out of two queues at the same time. This helps untangling rules a lot. Also, CRS is able to better optimize the data it needs to retrieve.
- If you have multiple branches that potentially can apply to the same item, write the ones with the highest resulting score first. Thus, the item will stop on the highest possible score that applies.
- If you change the score of a decision, verify if the rule ordering needs to be changed.

Match rates and results

FraudOne engines (ASV, APIA, ...) return a verification result, like item ok, bad signature, amount out of range, rule failed, etc. Besides that, the match rate and matching objects are also returned.

The thing to keep in mind about match rates is that not every verification result has a valid match rate. Some verifications fail because the account can't be found, the signature or check stock reference is not on file, etc. These kinds of results don't have a match rate because there is no reference to compare against. Therefore, it is not advisable to base your rules on match rates alone. If you want to verify match rates, keep in mind that some results don't produce one.


Names and comments

Rule names and comment fields can greatly help keeping your rule set readable.

When you create a new rule, the decision at the end of it will automatically be given a name, which defaults to the next available rule number. My advice here is, to change that to a readable name which tells you something about what that decision does. The decision name is used in CRS traces and also shows in the item comment field when it is presented for the next decision. Having a meaning behind this name can greatly help the fraud analyst that gets the item for review to understand why it got there.

Even though the system does not enforce this, it is a good idea to have unique decision names. Even if two decisions do essentially the same thing, it is a good idea to be able to tell them apart when you try to backtrack which rule applied to a specific item.

Another good way to keep the rule set readable is to use the comment fields where they are provided. The comment fields will not show in the rule overview, but they can help you understand what you actually tried to accomplish when you review your rule set a year after you have initially written it.

 One comment field can cause some confusion. That is the comment field inside a target description (not decision). While all other comment fields in the rule description do not change the item, but are only there to make rules readable, the target comment does not describe the target. It is meant to change the item comment field when placed in a new queue by this target. Leaving this comment empty is actually not a bad idea at all, since the system will then fill it by default with the decision name, score and risk of the item.

Keep it simple

Simple things work more reliable.

This is also true for CRS rule sets. When designing and writing a new rule set, try to do it the simplest way possible. The bigger and more complicated a rule set gets, the harder it will be to understand and review it. The probability of errors will increase with the complexity of your design.

Before you opt for a complex way of doing things, like having a parallel setup, try to assess if the savings or possible performance increase really justify the increased complexity and additional time you need for chasing flaws in rule design.

Related conditions

Sometimes you need different conditions in the rule tree to use the same value to compare against. The same amount limit might need to be used for different queues, the same random value for QA verification, etc.

The easiest way to accomplish this is to define a configuration variable for this. Then you only need to use that variable in all the conditions that need to be synchronized. From now on you can change all conditions by only changing the variable value – they will always stay in sync.

Engine only workflows

You can roughly classify FraudOne workflows in two categories:

- Workflows using visual review clients.
- Workflows using engines only, where FraudOne only provides a result, score and risk to a customer specific review system used to present items to fraud analysts.

While the examples shown so far focus on the first variety, since that is what any reader can re-create with FraudOne components alone, some things need to be said about engine only workflows too.

Engine only workflows are essentially the same thing – they only lack visual review queues. Usually, all items run through all engines, then get scored and sent to OUTPUT for review by other systems. The question here is when to score and prioritize items. There is no need to prioritize items for internal queues anymore, since all items run through all engines and the order in which they do so is not important. Sorting items to be processed by automatic engines is only important if:

- You have a limited time frame and there is the risk that the engines will not have the time to process all items. In this case you'll want the high-risk items clear the engines first.
- Your system is a 'run through' kind of system. The OUTPUT does not collect all items first but sends them to the following application one by one. In this case it might be interesting to have the high risk items be processed and leave the FraudOne system first.

Items need to be scored and sorted for the human review and this is a step following the FraudOne engine only workflow. Thus, scoring and sorting needs to take place for the OUTPUT queue.

You can use the same methods we described above for the visual (VSV) queue and apply it to OUTPUT. This includes score and risk calculation (based on amount and/or VIP). The information that is contained in the putter file includes, but is not limited to:

- Final result (the result you set in the target that points to the OUTPUT queue).
The final result can be used for any purpose you chose, like:
 - Specify a decision for that item (pay, reject, review, ...).
 - Contain routing information for other review systems (like a queue or analyst number to use).

- Carry other meaningful information for non FraudOne applications.
- Item comment. You can set that in the target that points to OUTPUT or leave it empty. If empty, it will be set by CRS to contain:
 - The decision name that applied to this item and moved it to OUTPUT.
 - The score of that decision.
 - The calculated risk for the item, based on the score above.

Applications following FraudOne can use that information to sort and route items for visual review.

Reporting errors

The time might come when you have a problem and CRS does not behave the way you expect. Fortunately, Kofax is here to help you with it. This chapter summarizes what information you need to gather to streamline the error processing.

First try to isolate the item that generates the error. Write down the document reference number. If possible, run only that one item again (you can try doing a reset on the item). While you do that, turn the CRS engine to trace level 'trace' using your Server Monitor and capture the trace output the item produces. Also copy the display output lines that CRS writes when starting – they contain the version number of the components used.

Now open the CRS rules editor and export the active rule set as an XML file.

If you have access to the system database, print the content of the DEB.SC_WORKFLOW and DEB.SC_RESULT table for the item identified (after the error occurred). If not, contact your system administrator.

This should help you put together following information:

- Operating system and version.
- Database system and version.
- CRS component versions.
You can find this information in the first lines CRS displays when it is starting on trace level 'D = display'. Copy those lines from the Server Monitor log window via the copy and paste function.
- The active rule set as exported XML file.
Export this file from the CRS editor.
- The trace log from the Server Monitor on trace level 'T = trace' containing the complete processing of the one item in question, including any error messages shown.
- Database listings for the item in question, identified by its document reference number.
You can obtain those by running following commands against the FraudOne database:

```
SELECT * FROM DEB.SC_WORKFLOW WHERE DOC_REF_NO = 'YourDocRefNo'
```

```
SELECT * FROM DEB.SC_RESULT WHERE DOC_REF_NO = 'YourDocRefNo'
```
- Add a short, but complete description of the error and expected behavior.

Wrap up

There is no actual wrap up for this chapter, individual paragraphs are short and concise.

If you have any good practices information of your own, that you consider important and are willing to share with other FraudOne users, submit it to Kofax. We will gladly include additional points here.

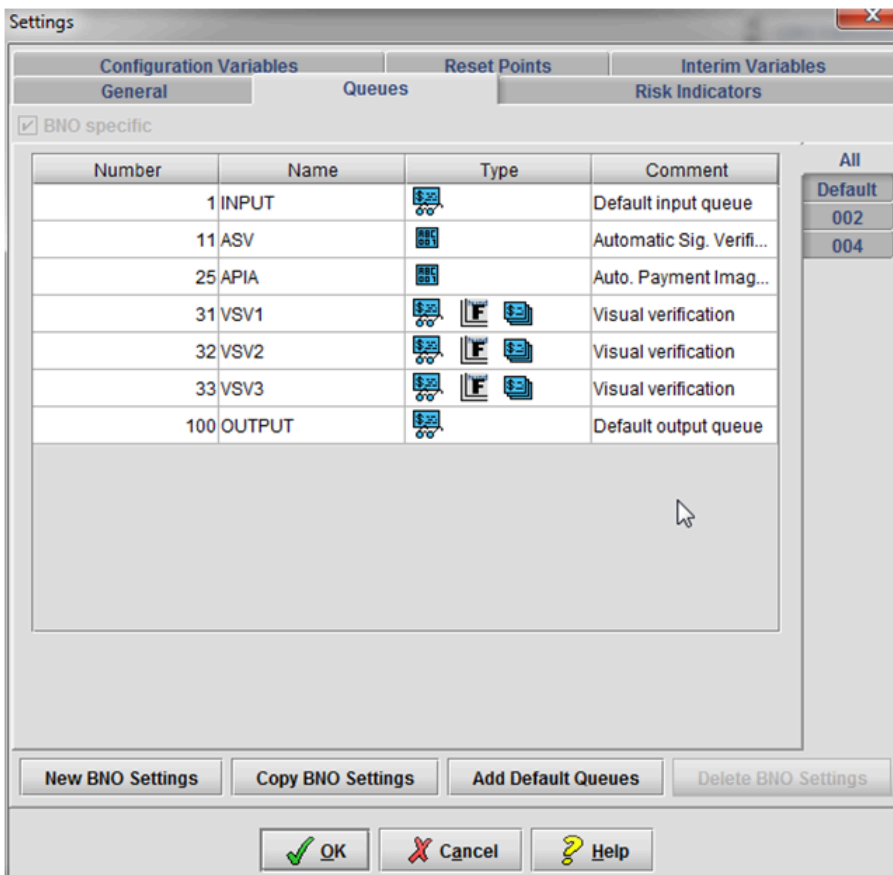
Chapter 12

Reference

Configuration elements

Queues

The CRS editor controls which queues will be available in the FraudOne systems. It also determines the way visual clients will display items while working on these queues.



The **BNO specific** check box controls if a unique queue list is used for all BNOs (not checked), or if each BNO has a queue list of its own (checked).

In case of a unique queue list, all BNOs use the same queue information, displayed in the **All** tab.

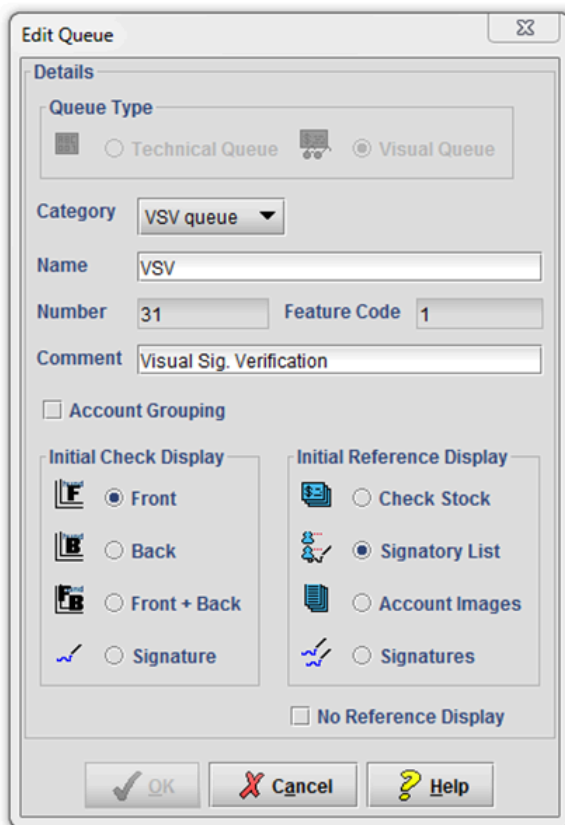
In case of BNO specific queue lists, the queue list used by visual clients will be determined according to the BNO number:

- If there is a tab (list) for the BNO in question, that list will be used.
- If there is no tab for the BNO in question, the list under **Default** will be used.
- In case the client works all BNOs (pseudo BNO 999), the **All** list will be used.

Non-visual clients, engines and CRS always work all BNOs and will use the **All** list.

If you create a queue in a BNO, it will be added to **All** automatically.

The queue definition also contains the parameters visual clients will use to display information. Thus, when you define a queue, you can control the way it will be shown in visual clients – no need to change client configuration. Non-visual components, like engines, will not use these definitions.



Queue Type	The type of the queue: Can be technical or visual. Technical queues are used by non-interactive components, like verification engines. Visual queues are used by verification clients that require human interaction.
-------------------	---

Category	The category (like VSV, VTV, ASV, APIA,...) of the queue Helps setting some defaults.
Name	The queue name. This will be displayed in visual clients when showing queue lists, selecting the queue to work on, etc.
Number	The queue identifier. This uniquely identifies the queue to the system. Even if two queues bear a different name, they are treated as one if the queue number is the same.
Feature Code	The feature code that identifies the result generated during processing on this queue. Other system components will use this code to search for the result.
Comment	A comment that helps you remember what this queue should be used for.
Account Grouping	This queue will use account grouping. Items will be presented to the visual clients grouped by account (if they are all in this queue). For more information consult the <i>Kofax FraudOne Administrator's Guide</i> .
Initial Check Display	The way the visual clients will display items while working this queue. Front - check front page. Signature only the signature area. Back - check back side. Front + Back - both check front and back.
Initial Reference Display	The way reference data will be displayed by visual clients. Check Stock - check stock display. Signatory List - signatures and payment rules for the account. Account Images - images stored on the account. Signatures - only signature display. No reference display - disables the area for reference information.

Two queues have a special meaning: INPUT is the input queue, where items enter the system. OUTPUT is the queue through which item will leave the system. The Putter will work the OUTPUT queue, retrieve the items and generate the output file. Both need to be present for the system to work.

You will need to stop and restart engines and servers for the change to take effect.

Risk indicators

Risk indicators are system information used by CRS to make decisions. CRS supplies a default list of indicators that can be extended if needed. Indicators need to be turned on if you want to use them. CRS will not retrieve data on indicators that are off. This helps optimizing data retrieval.

The default indicators (at the time this book has been written) are:

Location	Variable	Type	Id/ Feature	Source / Description
SignBase	Account type	Long	111 / 0	CUSTOMERTYPE in CUSTOMER table.
	Branch number	String	121 / 0	BRANCH_CODE in ACCOUNT table.

Location	Variable	Type	Id/ Feature	Source / Description
	ZIP code	String	122 / 0	New field DEMOGRAPHIC_LOCN in ACCOUNT table.
	Age of the account	Long	123 / 0	Calculated by the DBHelper process using the new 'account opening' date.
	Account free text	String	124 / 0	Account free text.
	Age of matched signature	Long	141 / 0	Calculated by the DBHelper process using DATESCANED of the matched signature.
	Customer since	Long	113 / 0	Calculated by the DBHelper process using the new 'Customer since' date.
	Age of matched signatory	Long	131 / 0	Calculated by the DBHelper process using the signatory's BIRTHDATE .
	Valued customer	Boolean	114 / 0	Retrieved by the DBHelper process using the customers VIP field.
Item (SC)	Last Queue	Integer	1 / 0	The queue number, the item comes from (the queue where the last result has been set).
	BNO	String	112 / 0	BNO in SC_INTERFACE table - bank number.
	Amount	Double Long (starting with release 4.4.1)	211 / 0	AMOUNT_LOCAL in SC_INTERFACE table - the amount used in internal processing.
	Form text code	String	212 / 0	Document type: FORM_TEXT_CODE in SC_INTERFACE table.
	Correction item	Boolean	213 / 0	Check if FORM_TYPE='3' in SC_INTERFACE.
	IRD Formtype	Boolean	214 / 0	Check if FORM_TYPE='4' in SC_INTERFACE - image replacement document.
	Document reference number	String	221 / 0	The document reference number uniquely identifying the item (from SC_INTERFACE).
	Item source	String	222 / 0	The source (capture) of the item (SC_INTERFACE).
	Serial number	String	223 / 0	The item serial number (SC_INTERFACE).
	Bankcode	String	224 / 0	Bank code (SC_INTERFACE).
	Presenting Bankcode	String	225 / 0	The bankcode of the presenting bank (SC_INTERFACE).
	Presenting Branch No	String	226 / 0	The branch number of the presenting branch (SC_INTERFACE).
	Transaction code	String	227 / 0	Transaction code (SC_INTERFACE).
	Form type	String	228 / 0	The item form type (SC_INTERFACE).
	Country	String	231 / 0	Country information (SC_INTERFACE).
	PN Date	String	232 / 0	Primanota date (SC_INTERFACE).

Location	Variable	Type	Id/ Feature	Source / Description
	Scan Date	String	233 / 0	Date the item was scanned (SC_INTERFACE).
	Proc Date	String	234 / 0	Item processing date (SC_INTERFACE).
	Customer No.	String	235 / 0	The customer number of this item (SC_INTERFACE).
	Account No.	String	236 / 0	The account number of this item (SC_INTERFACE),
	M Account	String	237 / 0	MICR line account number (SC_INTERFACE).
	Sign Date	String	238 / 0	Signing date (SC_INTERFACE).
	Clear Date	String	239 / 0	Clearing date (SC_INTERFACE).
	ASV result	Long	311 / 769	SC_RESULT - Automatic signature verification result.
Results (SC)	Age of matched signatory	Long	131 / 0	Age of the signatory that matched the item signature (from the SC_SIGNATORY table). Needs a good signature match.
	Age of matched signature	Long	141 / 0	Age of the signature that matched the item signature. Needs a good signature match.
	ASV result	Long	311 / 769	SC_RESULT - Automatic signature verification result.
	ASV match rate	Long	312 / 769	SC_RESULT - Automatic signature verification match rate.
	APIA result	Long	321 / 1025	SC_RESULT - Automatic check stock verification result.
	APIA match rate	Long	322 / 1025	SC_RESULT - Automatic check stock verification match rate (range 0 - 100).
	APIA raw result	Long	323 / 1025	SC_RESULT - Automatic check stock verification raw result.
	APIA raw match rate	Long	324 / 1025	SC_RESULT - Automatic check stock verification raw match rate, as provided by the engine.
	PAD	Boolean	331 / 513	SC_RESULT - Pre-authorized draft.
	PAD payor match	Long	332 / 514	SC_RESULT - Pre-authorized draft payor.
	PAD blacklist	Boolean	333 / 515	SC_RESULT - Pre-authorized draft blacklist match.
	PAD match rate	Long	334 / 513	SC_RESULT - Pre-authorized draft confidence level.
	PAD payor match rate	Long	335 / 514	SC_RESULT - Pre-authorized draft payor match rate.
	PAD blacklist match rate	Long	336 / 515	SC_RESULT - Pre-authorized draft payee name blacklist match rate.
	Getter result	Long	340 / 256	SC_RESULT - Getter processing (loading) result.

Location	Variable	Type	Id/ Feature	Source / Description
	Getter account missing	Boolean	342 / 257	SC_RESULT - Missing account information.
	Getter correction item	Boolean	344 / 259	SC_RESULT - Correction item.
	IRD detected	Boolean	345 / 260	SC_RESULT - Image replacement document.
	Getter Image missing	Boolean	346 / 261	SC_RESULT - Item image not available.
	Getter Image format bad	Boolean	347 / 262	SC_RESULT - Image format can't be processed.
	Getter Image size bad	Boolean	348 / 263	SC_RESULT - Invalid image size.
	Getter Image resolution bad	349 / 264	349 / 264	SC_RESULT - Invalid image resolution.
	Getter Signature size bad	Boolean	350 / 265	SC_RESULT - Bad signature snippet size.
	ATV result	Long	371 / 1537	SC_RESULT - Automatic technical verification.
	CAR/LAR result	Long	381 / 1540	SC_RESULT - Amount verification against check courtesy + legal amount.
	CAR/LAR match rate	Long	382 / 1540	SC_RESULT - Amount verification match rate against check CAR/LAR amount.
	Date (on check) result	Long	383 / 1541	SC_RESULT - Date range check.
	Date (on check) age	Long	384 / 1541	SC_RESULT - Check age in days or months depending on configuration.
	CAR/LAR compare result	Long	386 / 1542	SC_RESULT - Courtesy amount against legal amount comparison.
	CAR/LAR compare match rate	Long	387 / 1542	SC_RESULT - Courtesy amount against legal amount comparison match rate.
	CAR result	Long	388 / 1538	SC_RESULT - Courtesy amount verification result.
	CAR match rate	Long	389 / 1538	SC_RESULT - Courtesy amount verification match rate.
	LAR result	Long	390 / 1539	SC_RESULT - Legal amount verification result.
	LAR match rate	Long	391 / 1539	SC_RESULT - Legal amount verification match rate.
	Payee result	Long	401 / 1548	SC_RESULT - Payee verification result.
	Payee match rate	Long	402 / 1548	SC_RESULT - Payee verification match rate.
	Payee category	String	403 / 1548	SC_RESULT - The category of the payee that matched.

Location	Variable	Type	Id/ Feature	Source / Description
	Payee name	String	404 / 1548	SC_RESULT - The name of the payee that matched.
Other	Random value	Long	2 / 0	Random number between 0 and 100.

Depending on the specifications for your FraudOne release, this list may vary. See the corresponding *Kofax FraudOne Release Notes* and *Kofax FraudOne Feature Codes* for further information.

In case you need additional risk indicators, these can be added in two ways. For information located in specific FraudOne database locations, like SignBase extensions, or SignCheck results table, you can configure new risk indicators yourself. For other data you will need to contact Kofax to create a customer specific extension module that will provide the data.

Risk indicators that can be configured need to be defined in the `[VariableList]` section inside the `CRSDBHelper.ini` file in the configuration server. File names are case sensitive. The content of the section is described below:

Key	Description
VariableCount=0	The number of additional variables defined in the list. The default is zero also when no <code>CRSDBHelper.ini</code> can be found in the Configuration Database tables. Note that the variable definitions (see below) must be numbered sequentially from 1 to the number given in this configuration item. This numbering completes the Key Name by replacing the "<n>" with the specific variable number.
Var.<n>.Name=name	The name of the variable. This name is used only for display in the rules editor. Example: Var.1.Name=Customer Married Since
Var.<n>.Type=type	The data type of the variable. Supported values: int, double, bool, string. Example: Var.27.Type=double
Var.<n>.VarId=idnum	A unique variable identifier number. Numbers in the range 30000-30999 are available for customer use without reference to Kofax. Kofax reserve the right to use numbers outside this range at any time and for any purpose without further communication with the customer. Customers using numbers outside this range run the risk of causing unspecified system failures. Example: Var.12.VarId=30211

Key	Description
Var.<n>.Object	The database object type containing the data referred to by this variable. Useful values: random, result, extension0, extension1, extension2, extensionb random - Defines a variable that will contain a random number in the range defined by RangeLow and RangeHigh below. result - Defines a variable that can contain either a CRS Result value or a Match Rate value from a specific feature code. extension0 - Defines a variable that can contain a data item from the CUSTOMER EXTENSION table. extension1 - Defines a variable that can contain a data item from the ACCOUNT EXTENSION table. extension2 - Defines a variable that can contain a data item from the SIGNATORY EXTENSION table. extensionb - Defines a variable that can contain a data item from the ACCOUNT_IMAGE EXTENSION table.
Var.<n>.Link	For Object=result variables, this value identifies the Feature Code for which the result or match rate is required. For Object=extension2,-b variables, this value identifies the feature code to be used to select the matching signatory or account image.
Var.<n>.Match	For Object=extension2,-b variables, this value identifies which of the two possible matching object fields from the Feature Code record contains the SIGNO or IMAGENO.
Var.<n>.FieldId	For Object=result variables, this value selects either the Result (value 1) or the Match Rate (value 2) stored in the database for the Feature Code. For Object=extension0,-1,-2,-b variables, this value identifies the specific extension field from the corresponding table.
Var.<n>.RangeLow	For Type=int or Type=double variables, this value, if present, represents the minimum permitted value of the variable. This only affects the Rules Editor. The variable value itself is not checked against this range.
Var.<n>.RangeHigh	For Type=int or Type=double variables, this value, if present, represents the maximum permitted value of the variable. This only affects the Rules Editor. The variable value itself is not checked against this range.

Configuration variables

Usually, conditions inside the CRS rule graphs compare a risk indicator against a numeric value. If these values need to be changed often, or if two, or more, values from different conditions are related and need to have the same value all the time you can replace it/them with a symbolic variable. The actual value the variable takes can then be defined outside the rule graph in the CRS configuration.

Variables are defined in the **Configuration Variables** tab in the CRS properties window.

The variable will have a default value that the system will use if no value has been defined. Variable values will be defined in the CRS configuration (file crs.ini in the Configuration Server). The variable can be defined by BNO. The actual value used will be determined in this precedence:

1. The value defined as <variable name>=<value> in the section named [BNO-xxx] where xxx is the BNO number.

2. The value defined as `<variable name>=<value>` in the section named [Constants] if it is not defined for the BNO in question.
3. The value defined as default during variable definition in the CRS editor.

You will need to stop and restart the CRS engines for a change to take effect.

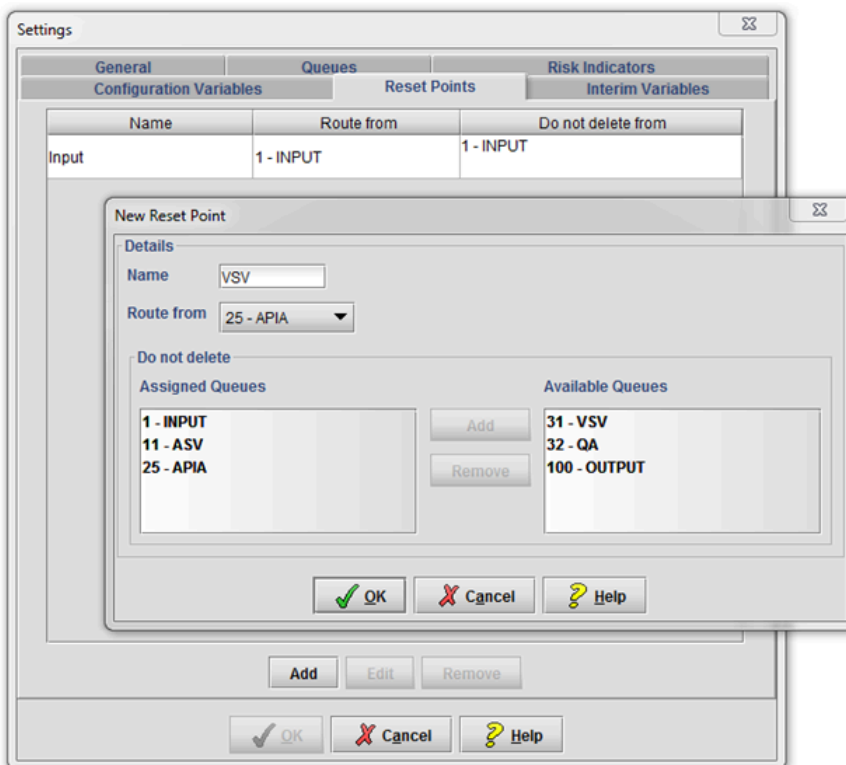
Reset points

Items where a processing error occurred, or a bad decision has been made can be reset in the FraudOne client, as long as the processing of the item has not completely finished and the item has not been removed from the system by the putter. In previous versions, resetting an item meant that all internal results of that item were deleted and the item started again from the INPUT queue.

Currently administrators can define arbitrary reset points. If a user makes a bad decision, it is not necessary to compute all automatic engine results for that item again. Therefore, it might not be necessary to have the item start from INPUT again, but maybe from the visual queue, where the bad decision has been made.

To define a reset point you must define the queue where the routing starts again (not the queue where the item will end up active, but the queue the item will be routed from). You also must define the results that the item should keep. For the example mentioned above you can start routing again in the APIA queue (last queue before VSV) and keep the engine results (ASV and APIA).

A reset point with the according data is shown below:



Name	The name of the reset point, as it will show up in the client.
Route from	The queue the routing will start in. This is the queue that will be shown in the 'Last queue' risk indicator, not the queue the item will end up after the routing. The queue used has to be one the item has previously passed through.
Do not delete	The results that should not be deleted when the item will be reset. You have to carefully consider which results to keep and which to reset. All results of the queues that will not be reset have to be kept, while all results of the queues that will be reset have to be deleted as well. An item cannot have duplicate results. Assigned queues have to be selected from the list of available queues on the right.

Interim variables

Interim variables are defined only by CRS and used to document a specific CRS state. They are not set or changed outside the CRS environment.

Interim variables are defined in the CRS settings and can be set in any CRS decision block. Variables can be queried in CRS conditions, shown in visual clients and exported by applications like the putter.

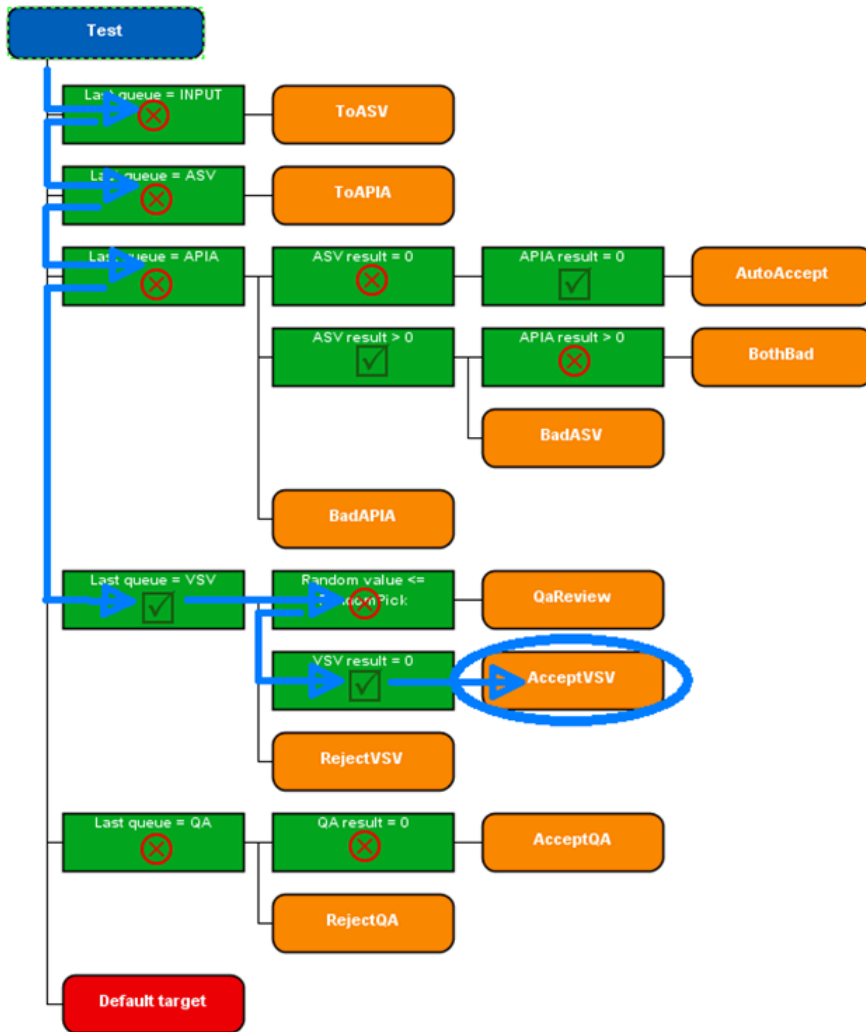
Interim variables are used to document complex situations that should not be recalculated every time. One common use is to determine and later query the completion of a parallel processing branch.

Rule elements

Rule graph

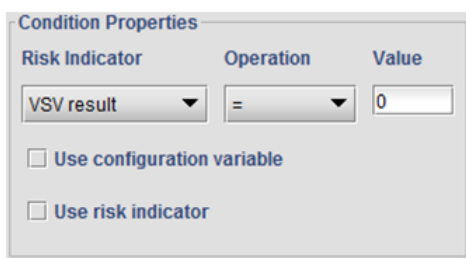
The rule graph is a decision tree that determines how an item will be processed. CRS will parse the tree from top to bottom, left to right and will try to reach the first decision element possible.

Tree branches contain conditions and are terminated by one decision. The decision at the end will be reached if all conditions on the path leading to it are true.



Conditions

A condition is an element that compares a system data value (risk indicator) against a given value or constant (also named variable) or another risk indicator. It is true if the comparison is true.



Depending on the type of the risk indicator, the comparison will be made as shown below:

Type	Comparison	Operator	Comment
last queue	range	=	Queue number.
long	range	>=	
		<=	
		=	
		between	Range between a and b.
		n/a	Not available. True if the risk indicator is undefined.
		available	Available. True if the risk indicator has been defined.
boolean	true / false	equals	true false
		n/a	Not available. True if the risk indicator is undefined.
		available	Available. True if the risk indicator has been defined.
double	range	>=	
		<=	
		=	
		between	Range between a and b.
		n/a	Not available. True if the risk indicator is undefined.
		available	Available. True if the risk indicator has been defined.
string	Regular expression	equals	Compared against a regular expression. True if the regular expression applies, false otherwise.
		n/a	Not available. True if the risk indicator is undefined.
		available	Available. True if the risk indicator has been defined.

Decisions

Decision elements can be found at the end of every CRS rule branch. They determine what needs to be done with an item reaching it.

The decision element itself contains information on item scoring:

Decision Properties

ID

Comment

Score

Weight None Amount VIP

Targets

Queue	Result	Type	Comment
31 - VSV		Above	
100 - OUTPUT	0	Below	

Interim Results

Interim Variable	Value	Comment

Menu entry	Description
ID	The decision name. This will be used to identify this decision.
Comment	The description for this decision. This will not be used by the system.
Score	The decision score. This will be used to calculate risk and priority.
Weight	The function used to determine item weight. This can be: <ul style="list-style-type: none"> • None - Item weight is considered to be '1'. All items are equally important. • Amount - The amount is used to weight items. The higher the amount, the more important the item is considered to be. • VIP - Very important person (or customer, or business). Accounts marked as VIP are considered more important than others. • Amount and VIP - Both amount and VIP are considered.
Targets	A list of targets for this item. The item (or rather a reference to it) will be moved to the queues given in the list of targets. Be careful when using more than one target. Consider reading chapter Doing it as a parallel process .
Interim Results	Interim variables that will be set in this decision. Interim variables can be used to record a CRS state.

Item risk is re-calculated each time the item reaches a decision. The formula used to compute item risk is:

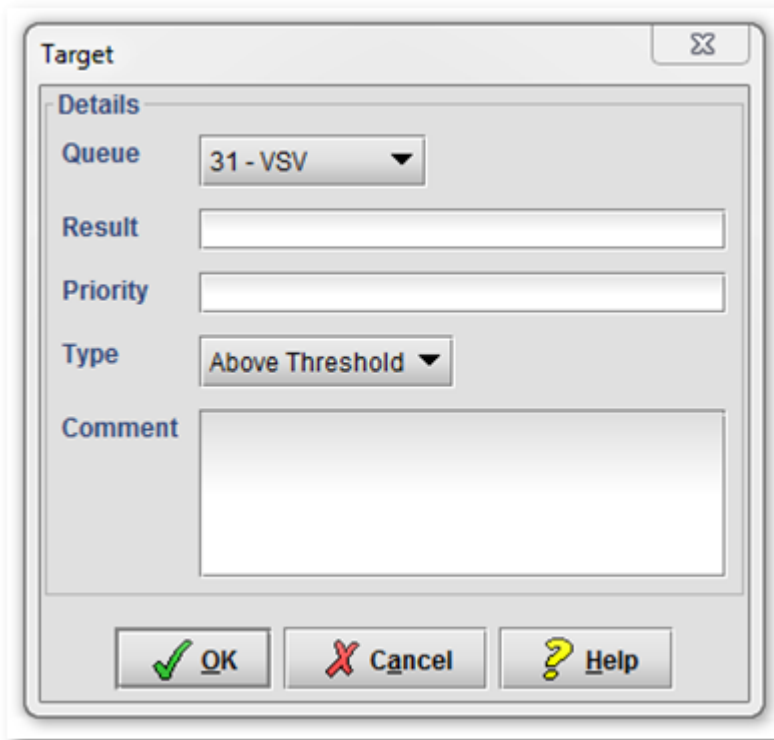
$$\text{risk} = \text{score} * \text{weight}$$

or

$$\text{risk} = \text{score} * \text{VIPmultiplier} * \log(\text{amount} + 10)$$

Where `VIPmultiplier` is configured in the CRS settings and is used if the item belongs to a VIP account.

The targets in the list look like this:

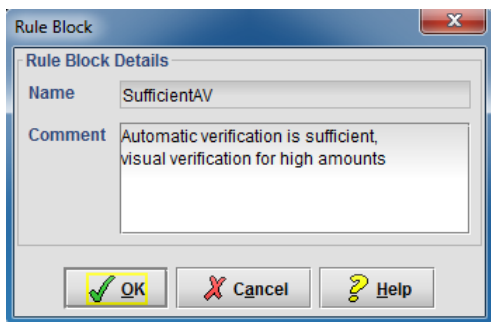


Menu entry	Description
Queue	The queue where the item will be moved.
Result	The new result to be initially used by the item. If left empty, this will be the result of the current operation. Usually needed to set the final result for the OUTPUT queue.
Priority	Fixed item priority for the queue above. If left empty, the priority is calculated based on the risk defined above. The higher the risk, the lower the priority. The item with the lowest priority value is processed first.

Menu entry	Description
Type	<p>Target type. Defines when this target will be applied:</p> <p>Always - This target always applies.</p> <p>Above threshold - This target only applies to items with a risk above the global risk threshold. Useful for volume management. By defining a global risk threshold in CRS configuration, you can control the routing of items based on their risk.</p> <p>Below threshold - This item applies to items with a risk below the global risk threshold.</p> <p>Below continue - This target applies to items with a risk below the global risk threshold. It is actually not a target, but tells CRS to continue parsing as if the decision would not exist (could not be reached).</p> <p>Wait - This target does not move the item to another queue. The item stops here. Usually needed for parallel processing, can also be used for dead ends – additional output queues.</p>
Comment	<p>Comment to be written into the database while routing the item. This is valid until the item is processed in the next queue. This has no descriptor function, but changes system data. If left empty, the system will create one based on following template:</p> <p>"<Decision ID>(<Score>:<Risk>)"</p>

Rule blocks

Rule blocks represent a continuous branch of the rule elements which can be used as a single CRS element and referenced from several parts of the rule tree. The content of a rule block (tree structure of rule elements) can be seen when it is expanded. Besides this a rule block contains the following properties:



Name	Rule block name, used as rule block's identifier.
Comment	Text field with an arbitrary description.

Default target

The default target is an error correction mechanism. Every rule graph has one at the end of it. It cannot be removed. If CRS parses an item and all rules in the rule graph do not apply, the default target will catch the item.

By itself, it is nothing more than a target of the type **Always**. If your rules contain an error and none of them apply to an item, the default target will move the item to the destination given.

You need to edit the default target before a new rule set can be saved.

See also [Planning for errors](#).