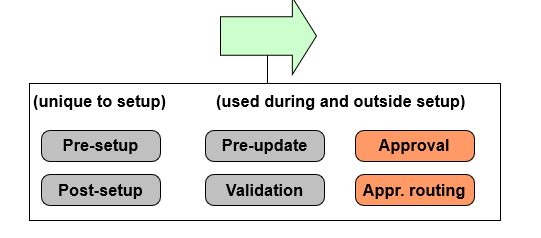
**Transaction Approval Rules**

**Review: Transaction rules**



* Approval rules determine if transactions (and/or checks) require approval
* If approval is needed, approval routing rules route approval activity to person whose approval is needed

Approval rules are run when a transaction is created and when it is edited or approved. (A transaction might require multiple approvals. Therefore, approval rules are run even when a transaction is approved.)

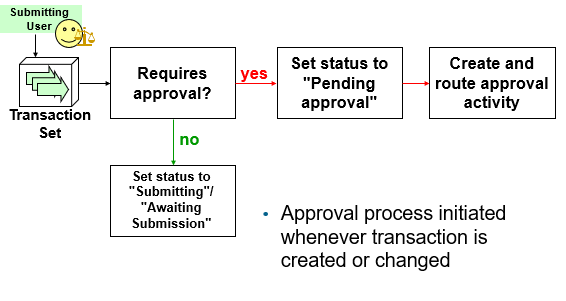
Approval routing rules are run whenever a transaction is designated as requiring approval.

There is no "Check approval" rule set. This is because the Transaction approval rule set is intended to cover both transactions and checks. Keep in mind that:

* + The root entity for all transaction rule sets is TransactionSet.
  + The TransactionSet entity has a CheckSet subtype.
  + The CheckSet subtype has a Checks array which points to the Check entity.

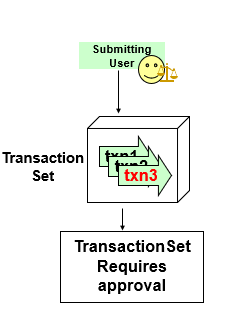
In other words, every check is associated with a transaction set.

**TransactionSet approval process**



Once a user submits (saves) a transaction, a decision process is initiated to determine if rules or authority limits require approval and if so, who should perform the first level of approval.

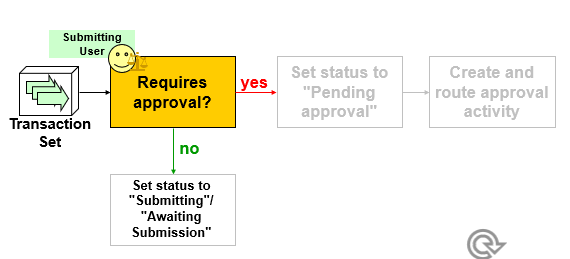
**Transaction approval**



* Transaction is always submitted by a user who may or may not have configured authority limits
* Each transaction in the set may be tested against some authority limit or business rule
* Approval always occurs at the TransactionSet level
  + Either all transactions in set are approved or none are approved
* The primary purpose of the transaction approval process is determine if a transaction can be acted on immediately or if it must be approved by someone of sufficient authority. In the base application, no other object requires this type of approval. This functionality is implemented only for transactions because the setting of reserves and execution of payments is typically one of the most tightly controlled activities within the claims process.
* Transactions are approved or rejected as a group because one transaction often influences another. For example, a single check might involve three payment transactions from two different reserve lines. If the check requires approval, for example, for a single transaction (txn 3 in the slide) because it is for a medical procedure and it is more than 25% over the standard amount for the specified procedure, then the check and all of its payment transactions must be set to pending. If approval was at the transaction level, then there would be ambiguity as to what to do with a check which consisted of two approved payment transaction and one pending approval payment transaction.

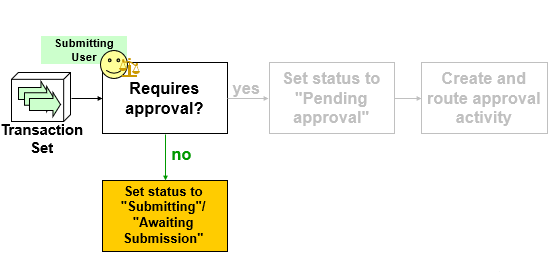
**Determining if approval is required**

* Two mechanisms for identifying transactions that require approval
  + Authority limits
  + Transaction approval rules
* Configure to use either or both mechanisms



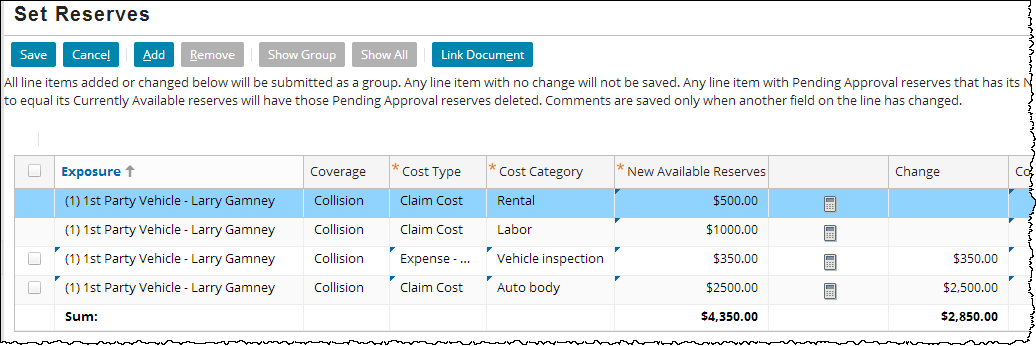
**If no approval is required**

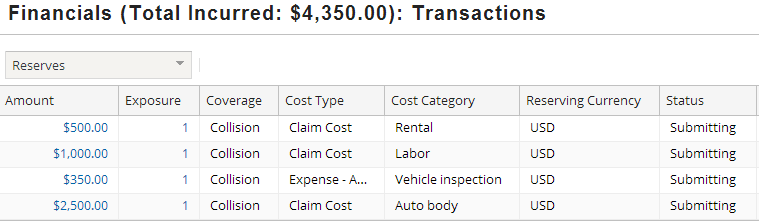
* For reserve transactions, reserve lines are created or modified
* For payment transactions, checks put into "awaiting submission" status



Recall that a check remains in "awaiting submission" status until the batch process which sends the check information to the check printing system. The next status it is set to is "requested" or "requesting", depending on the nature of the integration point to the check processing system.

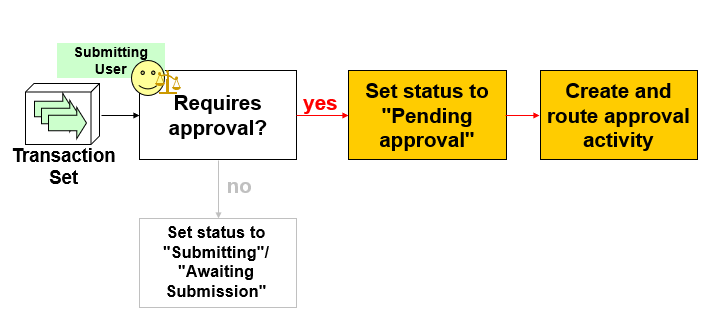
**Transaction set not requiring approval**





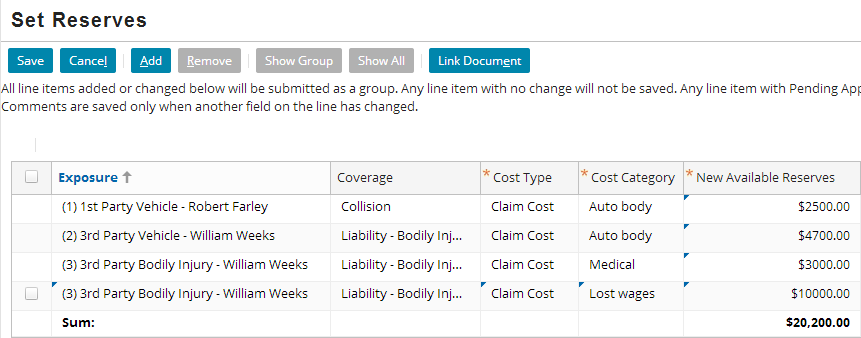
In the example above, a transaction set consisting of two reserve transactions is created. The transaction set does not exceed any authority limits or trigger any approval rules. Therefore, both transactions in the transaction set are listed as "Submitting".

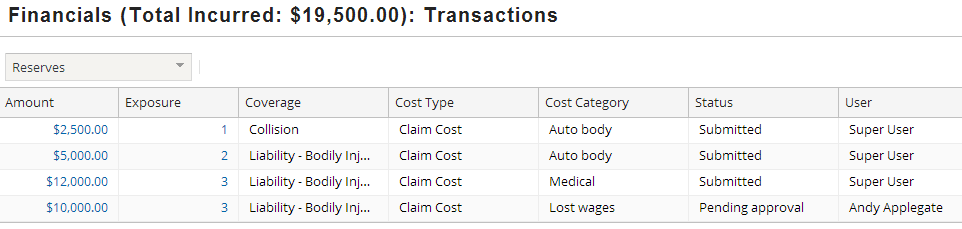
**If approval is required**



* TransactionSet and all transactions in it set to "pending approval"
* Special approval activity created and routed according to approval routing rules

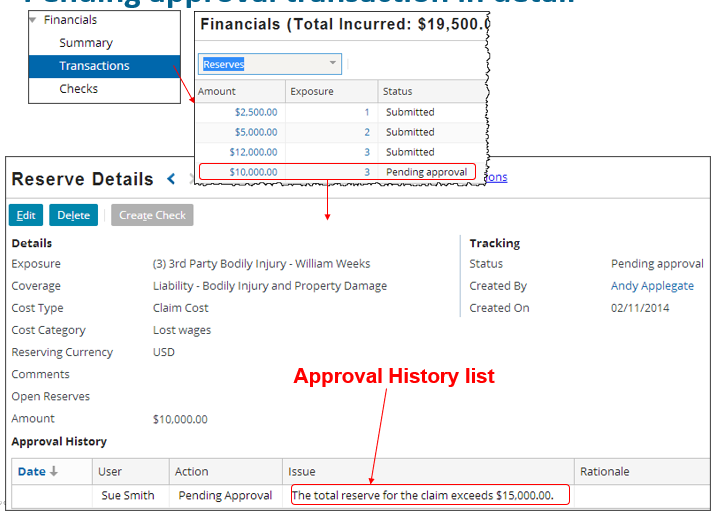
**Transaction set pending approval**





In the example above, a transaction set consisting of a single reserve transaction is created. The transaction set triggers the base application’s $15,000 authority limit for adjusters on creating reserves. Therefore, the transaction is listed as "Pending approval".

**Pending approval transaction in detail**

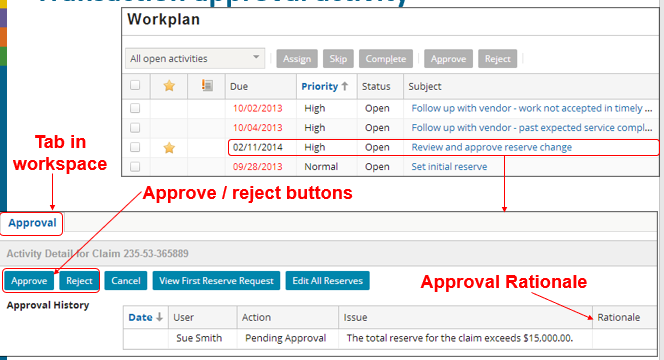


The adjuster can view detailed information about a transaction, including the reason for the approval, by clicking the link in the transaction list Amount column.

If multiple transactions are in a set they are shown separately.

The approval history of the transaction appears in the approval History list. Note the issue column, which includes “Issue” text from the transaction rule. This text would contain multiple causes if more than one transaction contributed to requiring the approval.

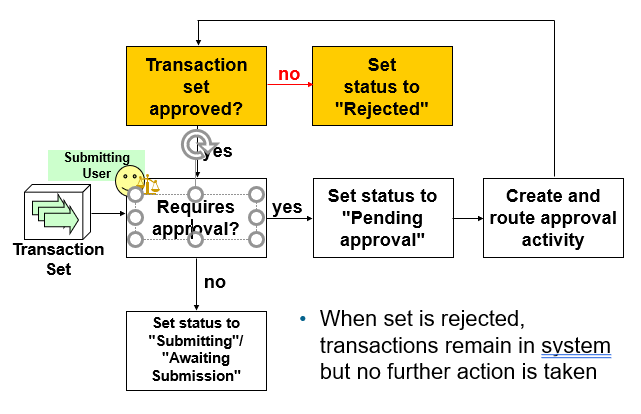
**Transaction approval activity**



Transaction approval activities are unlike regular activities in the following ways:

* + They are not closed by being skipped or completed. Instead, they are approved or rejected. The approval or rejection may include a “note” from the supervisor explaining the rationale for approving or rejecting.
  + They have toolbar buttons, list views, and a rationale field specific to the act of approving financial transactions.
  + They are routed not by activity assignment rules but rather by approval routing rules.

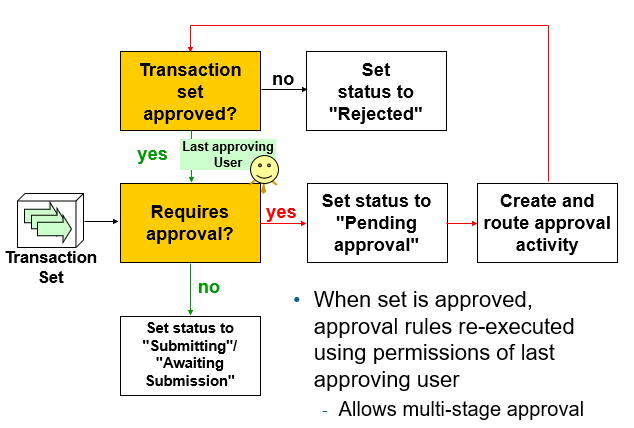
**Rejection of TransactionSet**



The first time the TransactionSet is submitted, it carries with it the submitting user.

Once routed for approval, the approving user is associated with the TransactionSet and her limits are used to determine if it requires approval again.

**Approval of TransactionSet**



As an example of multi-phased approval, a carrier might have a rule which says workers' comp checks over $10,000 must be approved first by the group supervisor and then by the regional manager. If this were the case, then you would expect to see an approval rule similar to the one shown below...

Condition: return transactionSet.Claim.LossType == LossType.TC\_WC and transactionSet.Subtype==“CheckSet” and transactionSet.ClaimAmount > 10000

Action: transactionSet.requireApproval("Workers' Comp check over $10,000 requires additional approval")

...and an approval routing rule similar to the one shown below...note that additional code is needed in the action to query for group name and user name (this will be discussed later in the lesson):

Condition: return(transactionSet.Claim.LossType == LossType.TC\_WC) and

transactionSet.ClaimAmount > 10000

Action: if (transactionSet.LastApprovingUser == null){

transactionSet.approveByGroupSupervisor() }

else  
 {

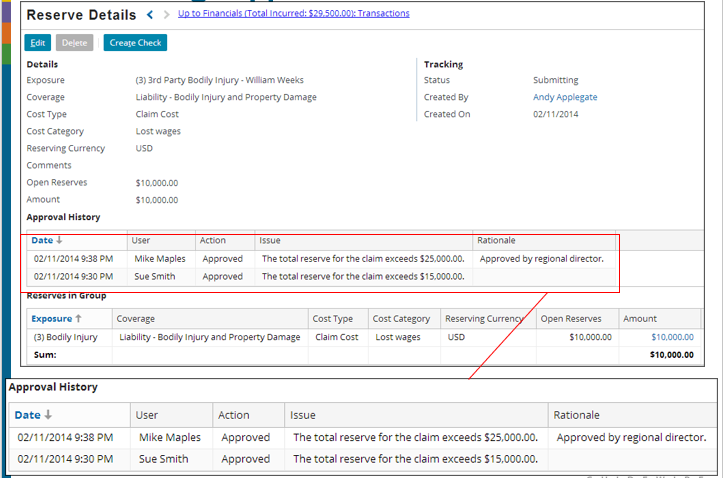
// code here to create Queries to retrieve User and Group

TransactionSet.setApprovingUser(

targetUser.AtMostOneRow, targetGroup.AtMostOneRow)

}

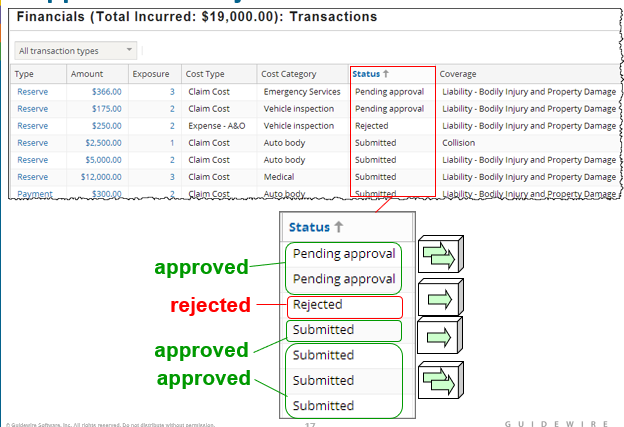
**Multi-stage approval**



In the example above, multiple issues (based on authority limits) prevent the transaction from being submitted, and multiple issues appear. After Sue Smith has approved, the transaction required further approval from Mike Maples (a higher authority limit had been reached). Mike entered a rationale when approving.

If multiple approvals are required by different users, then each approval appears as a line in the approval History list. Note that the user, date, action, and rationale for each approval event appears in the list.

**Approved and rejected transaction sets**

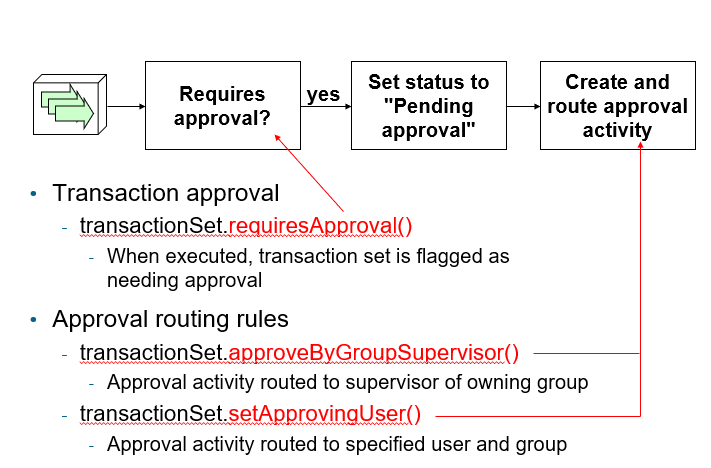


The example shown above shows four transaction sets. (You can identify which transactions belong to the same set only by viewing the details for each transaction.)

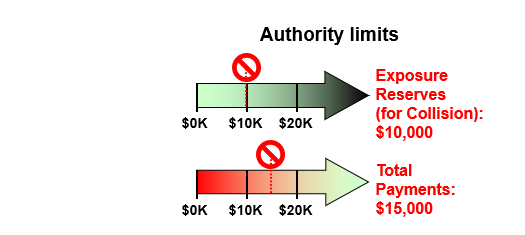
* + The first one (consisting of two reserve transactions) is waiting on approval.
  + The second one (consisting of one reserve transaction) required approval. It was not approved and went to rejected.
  + The third one (consisting of one reserve transaction) required approval. Once it was approved, it went to Submitting and is now Submitted.
  + The fourth one (consisting of three reserve transactions) required approval. Once it was approved, it went to Submitting and is now Submitted.
  + From the Transactions list, there is no status distinction between transactions which were immediately approved, transactions which were pending approval for authority limits reasons and then approved, or transactions which were pending approval for approval rules reasons and then approved. However, you would see differences in the Transaction Detail screen, as this lists the approval history of each transaction, if any.

Rejected transactions are obviously listed with a status of "Rejected".

**Primary transaction approval methods**

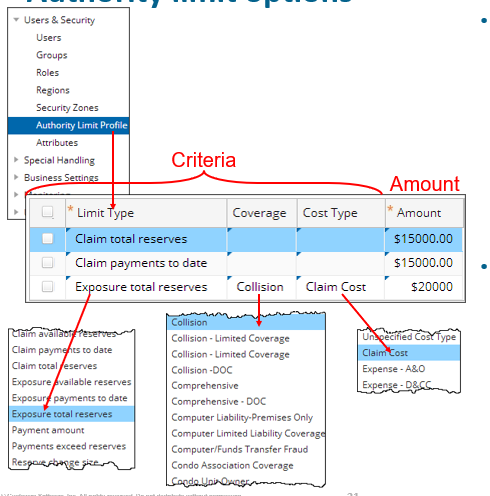


**Authority limits**



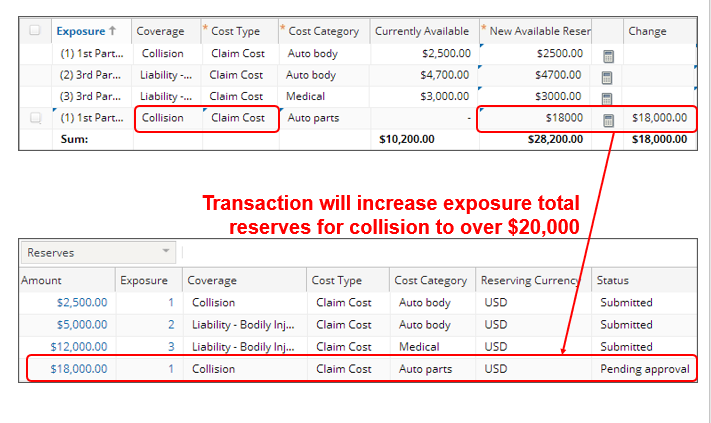
* An authority limit is a rule which limits the transactions an adjuster can create without approval
  + It consists of a criteria (such as "total payments for the entire claim") and an amount (such as $15,000)
* An authority limit is associated to a user through an authority limit profile. This is discussed in an upcoming slide.
* If an adjuster creates a transaction which meets the criteria of an associated authority limit, and the authority limit threshold is exceeded, then the transaction is immediately placed in a state of "pending approval" and an activity is generated and (typically) assigned to the supervisor of the group to approve or reject the transaction.
* Authority limits can be tied to any type of transaction. They can be tied to a single exposure, a single payment, the entire claim, or a change in the size of the reserve line. They can be associated to a specific coverage. They can be associated to a specific cost type.

**Authority limit options**



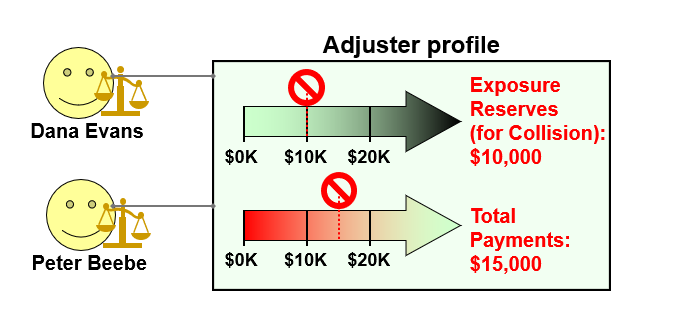
* Each criteria is a combination of:
  + A predefined limit type
  + An optional coverage
  + An optional cost type
* Each limit is a criteria plus an amount
* The options available in the Limit Type list cannot be modified: it maps to the AuthorityLimitType typelist, which is an internal (and therefore unmodifiable) typelist.
* The Coverage and Cost Type lists can be modified.
* The authority limit changes shown in the screenshot are made by a supervisor using the Administration screen > Users and Security > Authority Limit Profile.

**Exceeding an authority limit**



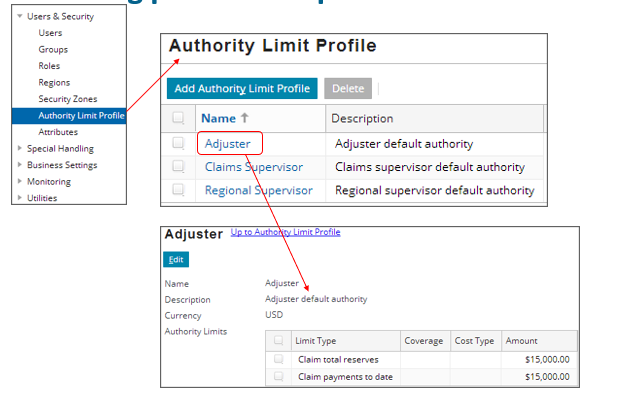
When a transaction exceeds an authority limit, the transaction is saved, but the status is set to "Pending approval". From a financials standpoint, no further action is taken on the transaction until it is approved.

**Authority limit profiles**



* An authority limit profile is a collection of authority limits assigned to one or more users
  + Can be pre-defined (created once and shared across multiple users)
  + Can be custom (created for a single user)
* An authority limit profile is a set of limits that specify threshold amounts for one or more types of transactions. When an authority limit profile is associated with a given user, if that user creates a transaction of a type listed in the profile which is at or exceeds the threshold amount, then the transaction is marked as requiring supervisor approval.
* In the example above, if either Dana Evans or Peter Beebe creates a reserve transaction for an exposure tied to collision coverage and the amount is over $10,000, then the transaction will require approval. Also, if either Dana Evans or Peter Beebe creates any type of reserve transaction that causes the total reserves for the claim to exceed $15,000, then the transaction will require approval.
* Technically speaking, there is an entity used by ClaimCenter for financials processing known as a transaction set. Every transaction belongs to a transaction set. Transaction sets can have one or multiple transactions in them. approval occurs at the transaction set level. If one transaction in the set requires approval, then the entire set requires approval. If the supervisor accepts the transaction, then the entire transaction set is accepted. If the supervisor rejects the transaction, then the entire transaction set is rejected. In practice, this level is typically transparent to an end user and is not discussed in detail here.

**Creating pre-defined profiles**

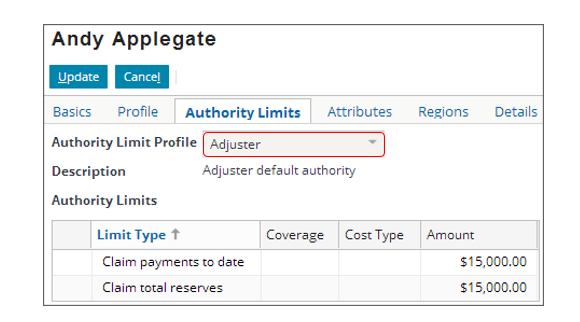


Pre-defined authority profile limits can be shared across multiple users. They are created and configured from the Administration tab's Authority Limit Profiles screen.

To configure an existing profile, click the profile name, click the Edit button, and add, remove, or modify limits as needed.

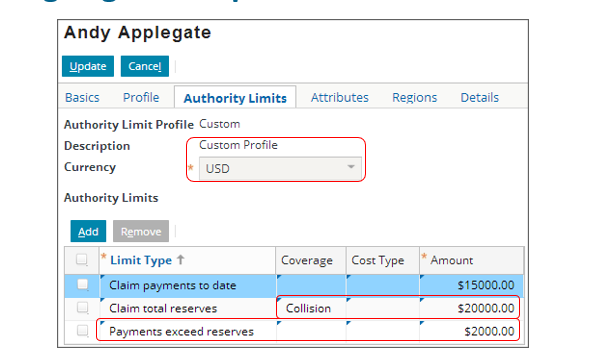
To create a new profile, click the Add Authority Limit Profile button. Then add limits as needed.

**Assigning pre-defined profiles to users**



* Each user can have at most one authority profile
  + Profile is assigned through the User screen
* To assign a pre-defined profile to a user, select the profile from the Authority Limit Profile dropdown. Note that when you choose a profile, the limits of the profile appear but are read-only. You cannot modify a pre-defined profile at the user level. You can modify it only at the profile level (which means the change you make will affect all users associated to that profile).

**Assigning custom profiles to users**



* Can also assign a custom profile to a user
  + Specifies set of authority limits that are not shared with other users
  + You can create a custom profile by selecting Custom from the Authority Limit Profile dropdown. Once you select Custom, the Authority Limit list becomes editable. Any custom profile created for a user is unique for that user. It cannot be applied to other users (unless you manually recreate it for the next user), and any changes to a given custom profile have no affects on any other profile.

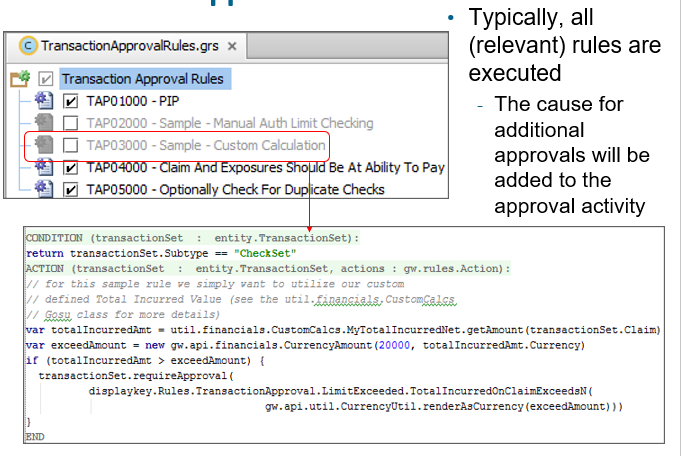
**Transaction approval, approval routing**

* Transaction approval rules implement approval logic that cannot be captured in an authority limit, such as
  + Logic beyond the given limit types
  + Logic tied to attributes other than coverage and cost type
  + Logic that cannot easily be associated to given users
* Approval routing rules assign approval ownership

Transaction approval rules implement approval logic that cannot be captured in an authority limit, such as

* + Logic beyond the given limit types
  + Logic tied to attributes other than coverage and cost type
  + Logic that cannot easily be associated to given users

**Transaction approval rules**

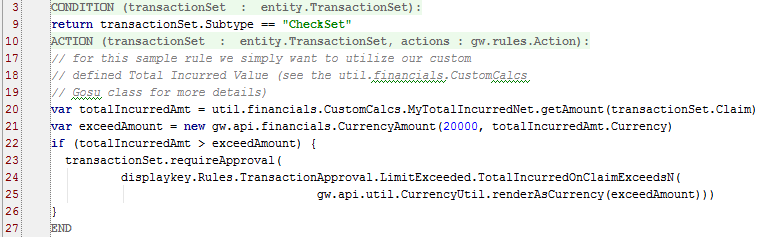


A transaction set either requires approval or does not require approval. However, there could be additional causes that the approving supervisor should know about. By executing all rules, the approval activity will contain a list of all limits that have been exceeded.

The disabled ruleset above is included as-is in the base application and is explained on the next slide.

**The requireApproval method**

* Syntax: *transactionSet*.requireApproval(*"reason"*)
  + Method only on transactionSet (not Transaction or Check)

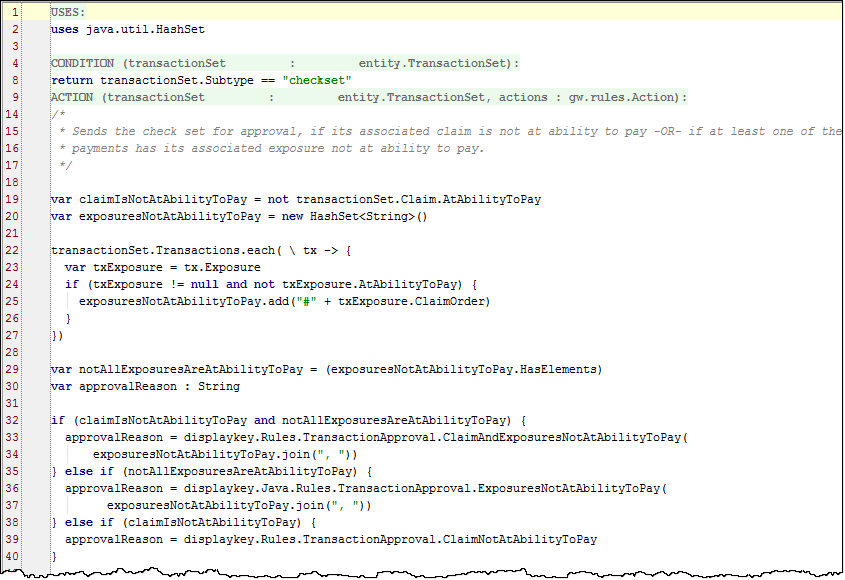


In the example above:

* + The condition checks to see if the transaction set is a check set.
  + In the rule actions, line 20 creates a variable and sets it to the total incurred amount of the claim.
  + Line 21 creates a variable and sets it to the exceeds limit value.
  + Line 22 checks to see if the total incurred amount is greater than the exceeds limit value.
  + If the limit is exceeded, Lines 23-25 identify that the transaction set requires approval and specifies the display key-supplied string to be used for the reason. This string appears both in the approval activity (which is seen by the supervisor) and in the transaction detail view (which any user with access to the claim can see by selecting the transaction from the transaction list).

**Approval Required if not at Ability to Pay**

****



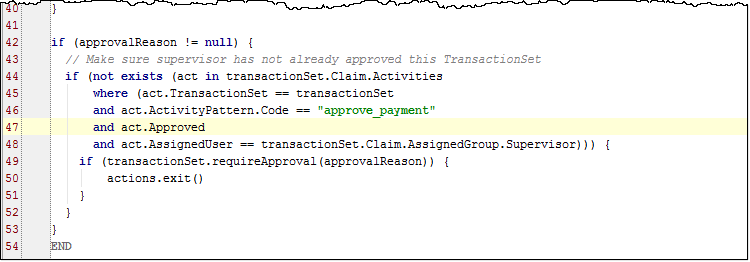
This rule is enabled in the base application and no code has been altered. One of the reasons for this rule is to designate *imported* checks for approval if the imported claim/exposures are not at ability to pay. This rule impacts the behavior of the webservice api ClaimFinancialsAPI.gs which performs such an import. Another case would be checks on Auto First and Final claims.

In the example above:

* + The condition checks to see if the transaction set is a check set.
  + Line 19 sets a boolean variable by using the Claim.AtAbilityToPay property to check if the claim is currently at ability to pay. No validation is performed, instead the value is checked.
  + Lines 20 – 27 sets a HashSet variable for use in determining if the array of related Exposures is at ability to pay using a similar property (Exposure.AtAbilityToPay).
  + Lines 29 – 40 set display key-supplied string values (approvalReason) for three specific cases depending on whether the claim or exposures are not at ability to pay. If one or more exposures is not at ability to pay, the exposure # is included in the “Issue” reason for approval.

**Approval Required if not at Ability to Pay (cont.)**

****



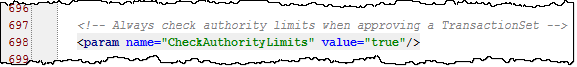
Lastly, the rule guards against re-approved already approved checks, in the case of approved checks at a lower claim/exposure maturity level.

Line 49 uses the requireApproval() method with the approvalReason string argument as defined previously.

**Manual evaluation of authority limits**

* You can perform manual authority limit checking for transactions by

1. Setting CheckAuthorityLimits to false in config.xml

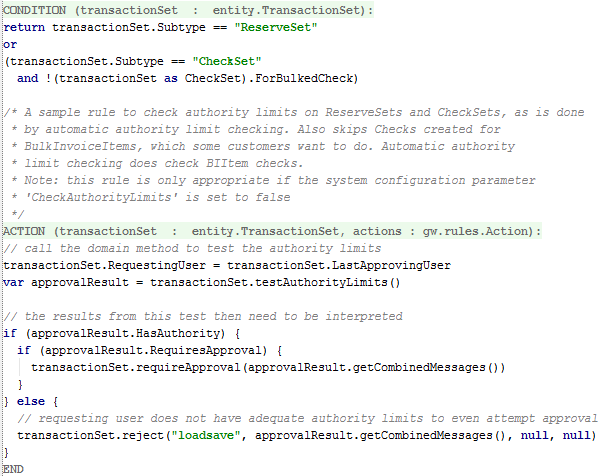


2. Writing selective rules that perform “manual” checks for users’ authority limits and reject or require approval

3. Writing default rules that perform checking against limits for all other transactions

**Manual evaluation of authority limits example**





The screenshot shown above is the Sample approval rule Manual Auth Limit Checking rule that is disabled in the base application (TAP02000). The example illustrates that some customers may want to configure the system to skip automatic authority limit checks for every check because they use the Bulk Pay feature and don't want these tests run on bulked checks.

1. To implement a rule like the example, you have to set CheckAuthorityLimits to false in config.xml.
2. This rule needs a condition that tests for the kind of transaction that you wanted to perform selective approval for, e.g. checks in bulk payments
3. If you want to check approval, you need to

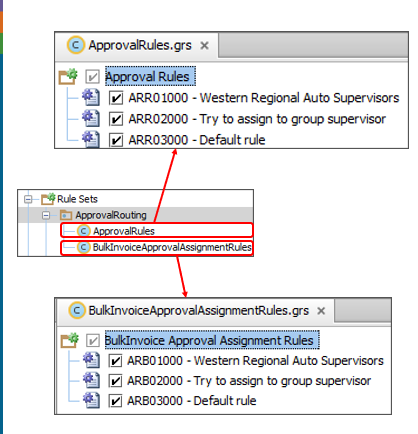
* (a) determine if any limits are configured that would apply
* (b) determine if they will be exceeded i.e. would the user fail for the current transactionset and if so
* (c) state the reason for the failure.

The object returned by TransactionSet.testAuthorityLimits() (for example, approvalResult) has 3 properties than can be used to do this:

* HasAuthority (boolean) true if limits do exist for this type of transaction
* RequiresApproval (boolean) true if the transaction is more than the limit
* CombinedMessages (string) contains the list of reasons it failed

Note that if there are no limits configured for a user, you probably don't grant approval. If there are limits and they pass, then you fall through to the next rule or approval is simply not required.

**Approval routing ruleset category**



* Approval routing akin to assignment for approval activities
* One set of rules for transaction set approval routing, and one for bulk invoice approval routing
* Typically, all (relevant) rules are not executed
  + As soon as one rule makes assignment, execution exits out of rule set
* All rules shown above are in the base application. Once you identify and succeed in routing to a user, you want to exit the rule set to avoid another rule routing to a less appropriate user.

**Approval routing methods**

* Syntax: *transactionSet*.approveByUserSupervisor()
  + No parameters
  + Assigns to group supervisor based on the user who submitted the transaction
    - If submitting user is in multiple groups, goes to the supervisor of the owning group



The approveByUserSupervisor() method works directly with the group hierarchy and assigns the approval activity to the supervisor of the submitting owner. If that supervisor does not have authority, then when the approveByUserSupervisor() method runs again, it will assign the activity to the next supervisor up the chain.

•If the submitting user is in the owning group and is not the supervisor, then approval goes to the group supervisor.

•If the submitting user is the group supervisor, then ClaimCenter escalates the approval to the group supervisor of the next group up the group hierarchy.

•If the submitting user is a group supervisor someplace higher in the hierarchy, then the approval escalates to the supervisor of the next group up the hierarchy.

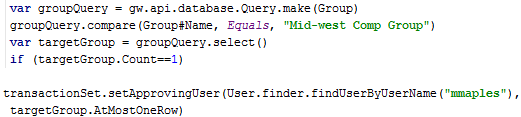
•If the submitting user is in one and only one group, then the approval goes to the supervisor of that group. Otherwise, approval goes to the supervisor of the owning group.

If ClaimCenter works its way through the various conditions to the supervisor of the root group without making an assignment, then the approval will be assigned to user *defaultowner.*

* Syntax: *transactionSet*.approveByGroupSupervisor()
  + No parameters; routed to supervisor of owning group
  + Returns false if user needing approval is group supervisor



* Syntax: *transactionSet*.setApprovingUser(*user*, *group*)
  + Routed to specified user and specified group
  + Use Queries and/or findUserByUserName() to retrieve users and groups



The approveByGroupSupervisor() method returns a boolean. The value is true if the user who initiated the action that requires approval is not the group supervisor. The value is false if the user who initiated the action that requires approval is the group supervisor.

• If the transaction set has transactions associated with only one exposure—the method assigns the approval activity to the supervisor of the group assigned to the exposure.

• If the transaction set has transactions associated with more than one exposure—the method assigns the approval activity to the supervisor of the group assigned to the associated claim.

In either case, if the supervisor of the group is the same as the user requesting approval, then the activity remains unassigned. *If you use this method, you must explicitly handle this case, most likely by using setApprovingUser(user, group).*

The example shown above for the setApprovingUser(user, group) method uses both a Query and the findUserByUserName() method. This may not be how you retrieve these values to pass in as arguments, but this is shown for instructional purposes so either could potentially be employed. Queries are discussed in the Configuration Fundamentals course, which is a pre-requisite to this course.

**Exiting out of approval routing rules**

* Once object is successfully assigned, you typically want no other rules to be executed
  + Approval routing methods are boolean – return true if the assignment was successful
  + Otherwise, fall through to a default assignment rule

