Working with the iWD Business Process in IRD

4/3/2026

IWDBP Strategies & Subroutines in 8.5.104

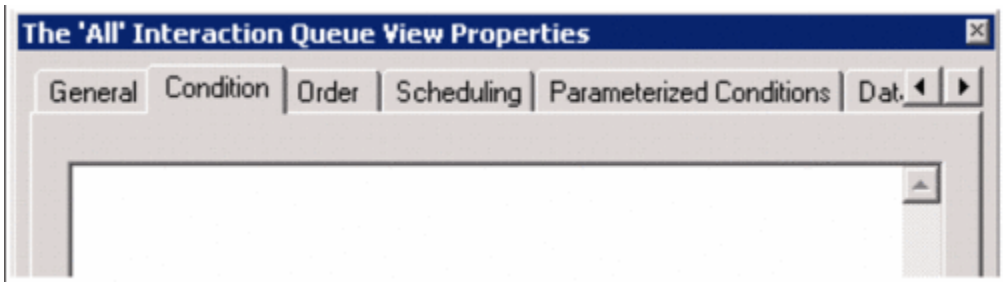
Classification Strategy

Classification Strategy

The purpose of this strategy is to invoke corresponding classification rules, analyze the result of the rules application and place the interaction into the appropriate queue, depending on the result.

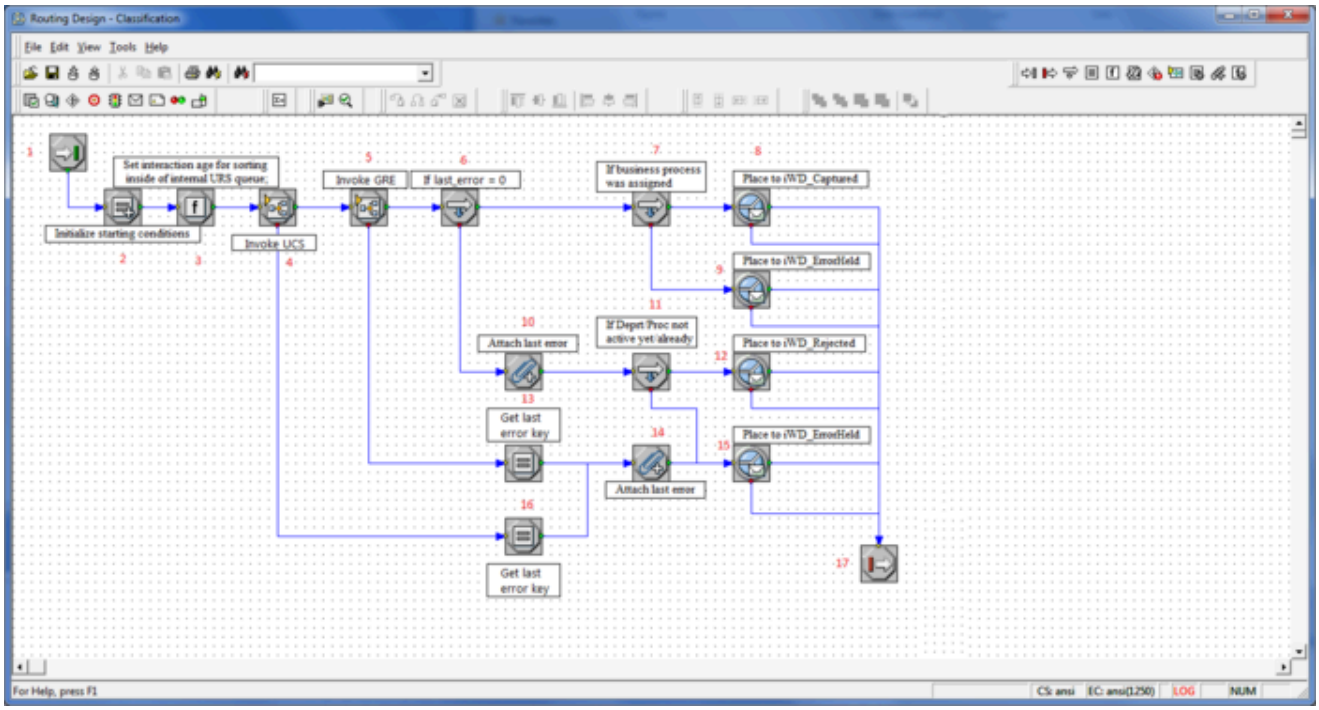
This strategy processes interactions from the following queues:

- `iWD_New`—Interactions have to satisfy the following conditions:
  - There are no conditions here.
  - Interactions are taken in order they were submitted.



Flow Summary

Click to enlarge.



Flow Detail

1. Entry to the Classification strategy.
2. A variable is initialized: `_delay_ms` specifies the delay (in milliseconds) between attempts to invoke rules.
3. A command is sent to URS to use interaction age while sorting interactions in internal queues.
4. The InvokeUCS subroutine is invoked to create new interaction in the UCS database.
5. The InvokeGRE subroutine is invoked.
6. Check if InvokeGRE subroutine finished without errors.
7. Verification is done to check if a business process was assigned by a classification rule.
8. If a business process was assigned, then the interaction is placed in the `iWD_Captured` queue.
9. If no business process was assigned, the interaction is placed into the `iWD_ErrorHeld` queue.
10. The last error is attached to user data as a key-value pair with the key `IWD_GRE_Error`.
11. A check is done to see if the error code is related to the iWD Department or Process not being available (for example, if the current date is outside of the Start and End Dates of the Department or Process).
12. If the Department or Process is not active yet, the interaction is placed in the `iWD_Rejected` queue.
13. Set `_last_error_key_` to `IWD_GRE_Error`.
14. The last error is attached to user data as a key-value pair with the key `_last_error_key_`.
15. The interaction is placed into the `iWD_ErrorHeld` queue.

16. Set `_last_error_key_` to `IWD_UCS_Error`.
17. Exit Classification strategy.

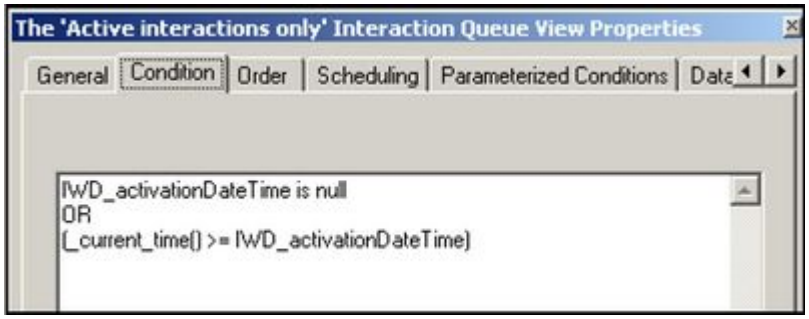
Prioritization Strategy

Prioritization Strategy

The purpose of this strategy is to invoke the corresponding prioritization rules, analyze the result of the rules application and place the interaction into the appropriate queue, depending on the result.

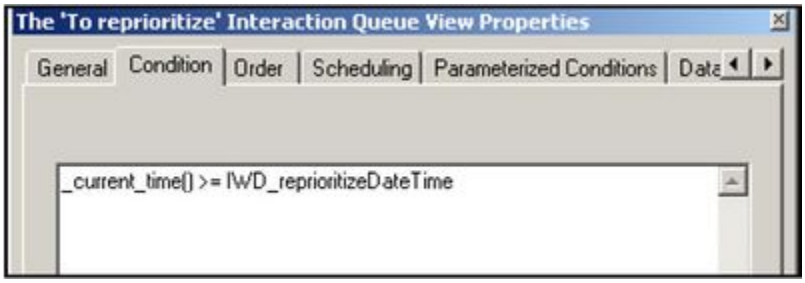
This strategy processes interactions from the following queues:

- `iWD_Captured`—Interactions have to satisfy the following conditions:
  - Active interactions only (interactions which do not have the property `IWD_activationDateTime` set, or this property has a time stamp which is in the past.
  - Interactions are taken in the order they were submitted.



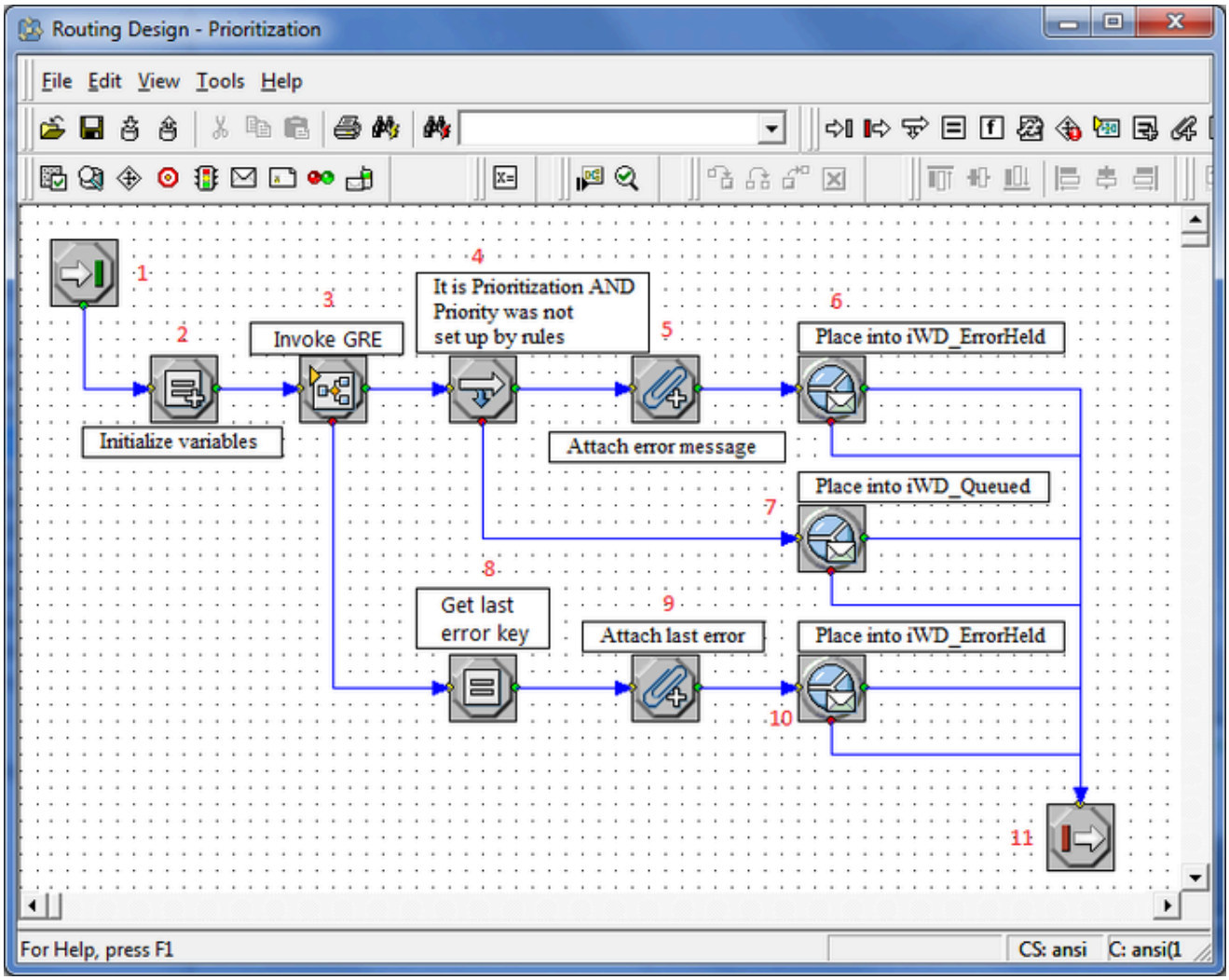
Active Interactions only

- `iWD_Queued`—Interactions have to satisfy the following conditions:
  - Interactions that are subject for immediate reprioritization (interactions that have the property `IWD_reprioritizeDateTime` set to a time stamp which is in the past)
  - Interactions are taken in order of `IWD_reprioritizationDateTime` (oldest first).



For reprioritization

Flow Summary



Flow Detail

1. Entry to the Prioritization strategy.
2. Variables are initialized:
   - `_source_queue` is the queue from which the interactions came. It will be used to determine if the prioritization service is being requested for initial prioritization or reprioritization.
   - `_error_timeout_ms` specifies the delay (in milliseconds) between attempts to invoke rules.

- `_default_priority` specifies the priority which will be assigned if a priority is not specified by the customer (as part of the task capture) or by rules.

3. The InvokeGRE subroutine is invoked.
4. A check is made to see if this is the first time that prioritization rules are being evaluated for the interaction, and the priority was not set up by any rules. If this check is false, flow goes to 7.
5. The error message `Priority is not set up by rules` is attached to Interaction Server data as a key-value pair with the key `IWD_Prioritization_Error`.
6. The interaction is placed in the `iWD_ErrorHeld` queue.
7. The interaction is placed in the `iWD_Queued` queue.
8. Set _last_error_key_ to `IWD_GRE_Error`.
9. The last error is attached to user data as a key-value pair with the key `_last_error_key_`.
10. The interaction is placed in the `iWD_ErrorHeld` queue.
11. Exit Prioritization strategy.

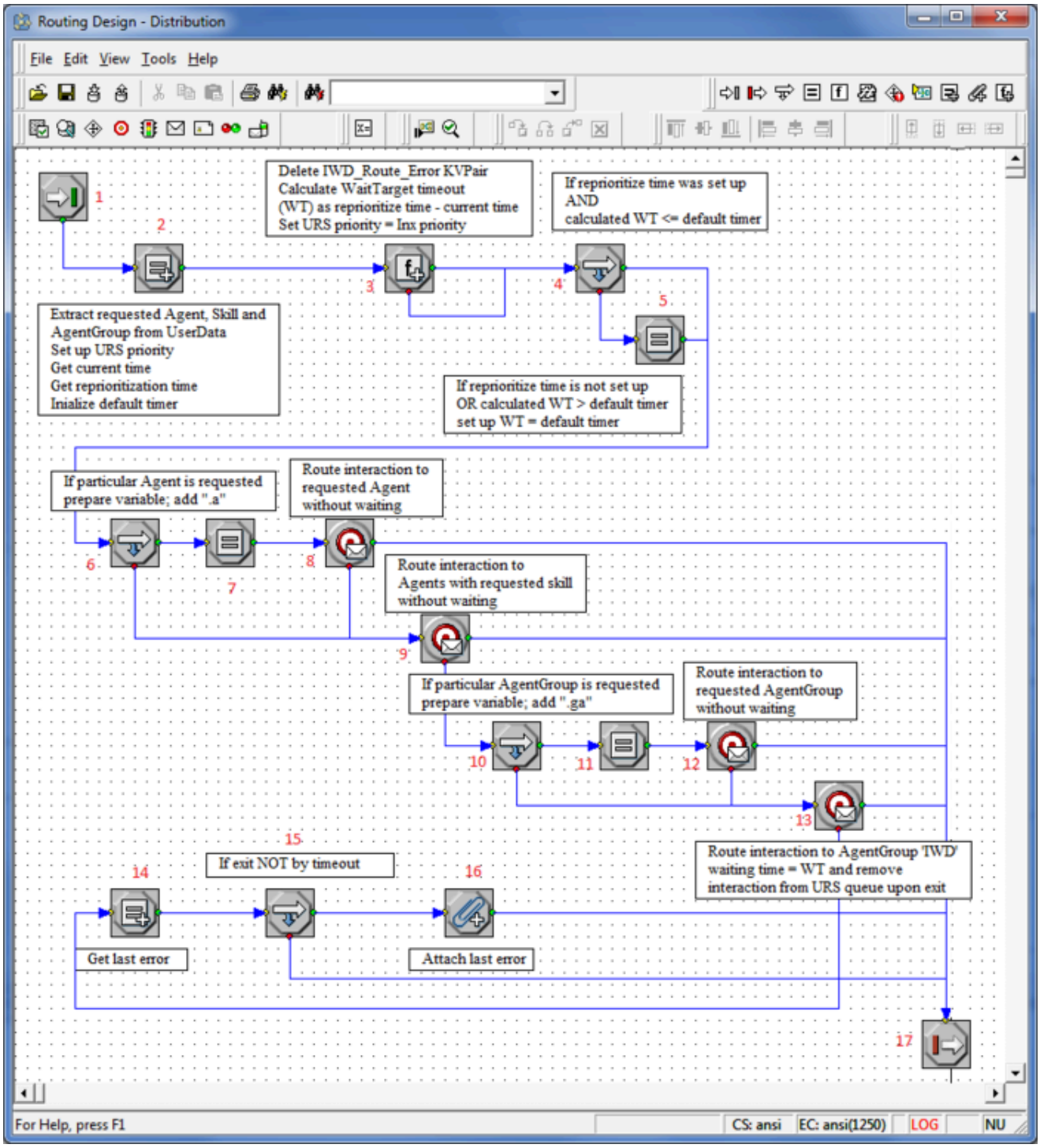
Distribution Strategy

Distribution Strategy

This strategy routes interactions to a requested Agent, requested Agent Group, requested Skill, or to the default iWD Agent Group. This strategy processes interactions from the following queues:

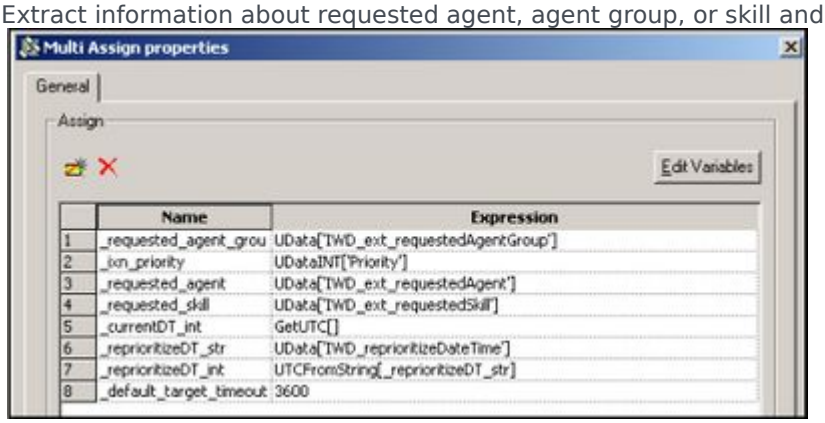
- `iWD_Queued`—Interactions have to satisfy the following conditions:
  - Interactions that are not subject for immediate reprioritization (interactions that do not have the property `IWD_reprioritizeDateTime` set, or that have this property set to a time stamp that is in the future).
  - Interactions are taken in order of priority (highest priority first)

Flow Summary



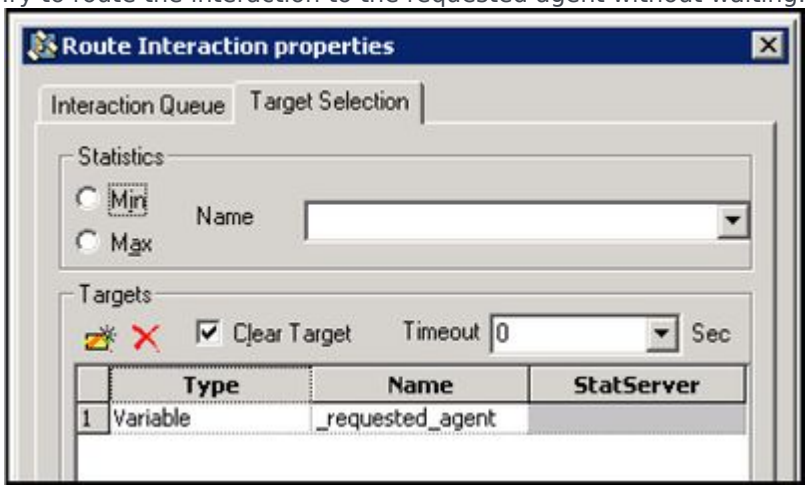
Flow Detail

1. Entry to the Distribution strategy.
2. Extract information about requested agent, agent group, or skill and initialize internal variables.



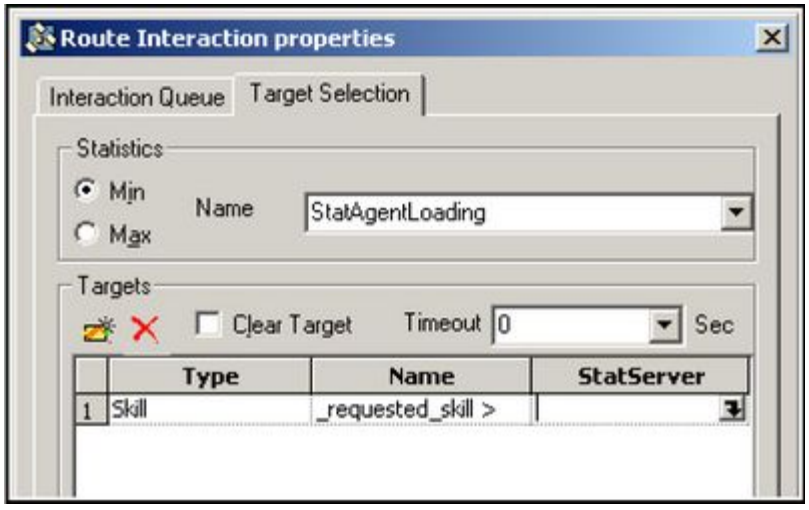
Multi-Assign - Requested Agent and Skill

3. A calculation is done to determine the timeout—how long the interaction should wait for a target to become available.
4. If the reprioritize time was set up and the calculated timeout is less than or equal to the default timeout (`1 hour`, see Step 1), then the timeout remains as it is.
5. If the reprioritize time was not set, or the calculated timeout is greater than the default timeout, then the waiting timeout is set to the default (`1 hour`).
6. Analysis is done to determine whether an agent was requested.
7. If an agent was requested, the URS variable is prepared (`.a` is added).
8. Try to route the interaction to the requested agent without waiting.



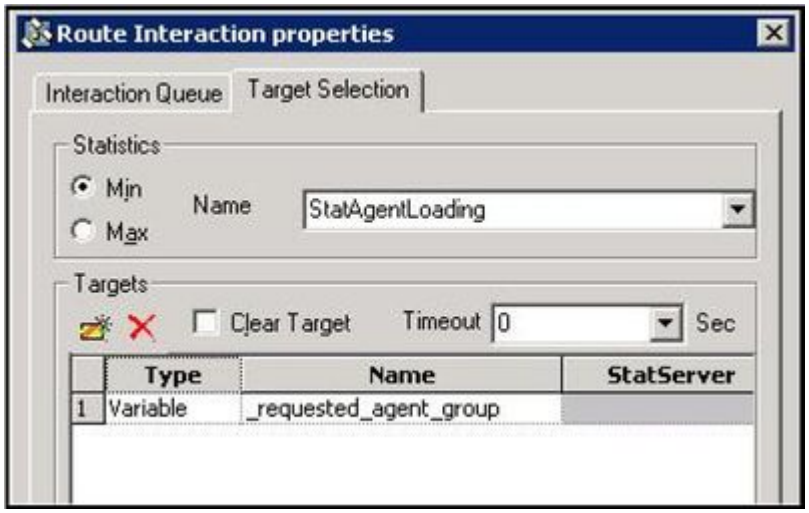
Route to agent

9. Try to route the interaction to an agent with the requested skill without waiting.



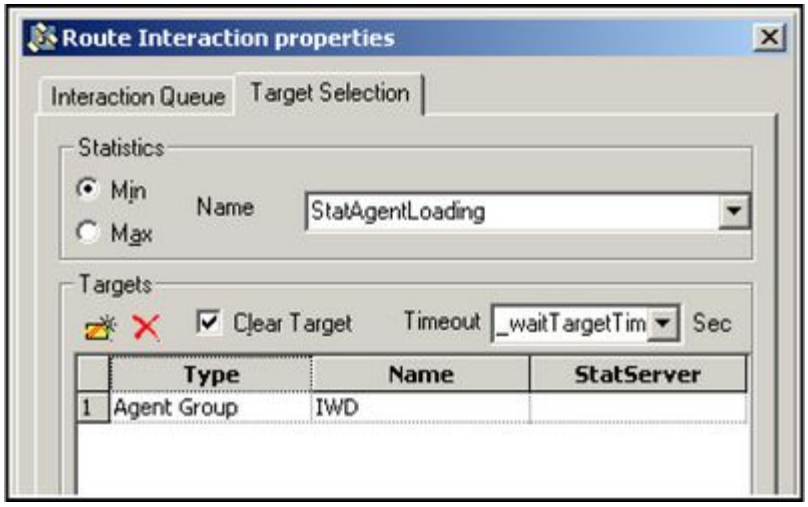
Route to Skill

10. Analysis is done to determine whether an Agent Group was requested.
11. If an Agent Group was requested, the URS variable is prepared (`.ga` is added).
12. Try to route the interaction to the requested Agent Group without waiting.



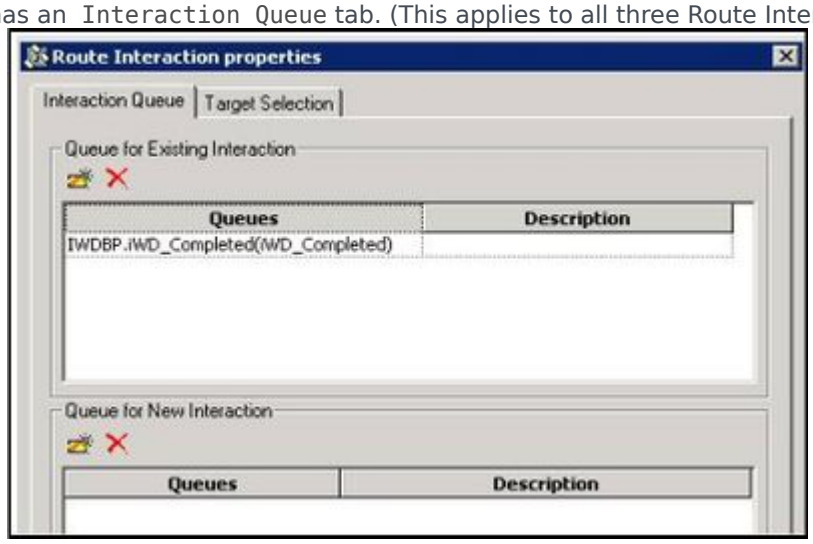
Route to Requested Agent Group

13. Try to route the interaction to the iWD agent group with a wait time of 60 seconds.



Route to Agent Group

14. Get the last error.
15. Verification is done as to why the target was not found.
16. An error code is attached in case of any error other than a timeout. If more than one target is available, URS uses the `StatAgentsLoading` statistic to select the Agent who has the minimum load (this applies to routing to Skills and routing to Groups only; routing to a requested Agent does not use statistics). For more information about this statistic, see the Universal Routing 8.1 Reference Manual. The Route Interaction object also has an `Interaction Queue` tab. (This applies to all three Route Interaction objects in this strategy.)



Route Interaction Properties—Interaction Queue

The optional `Interaction Queue` tab enables you to specify two types of queues:

- Queues for existing interactions (the queue in which the interaction should be placed after the agent is done working with it).
- Queues for new interactions (the queue in which new interactions created by the agent should be

placed).

A Description (optional) appears as a hint on the agent desktop as to where to place the interaction.

17. Exit from the Distribution strategy.

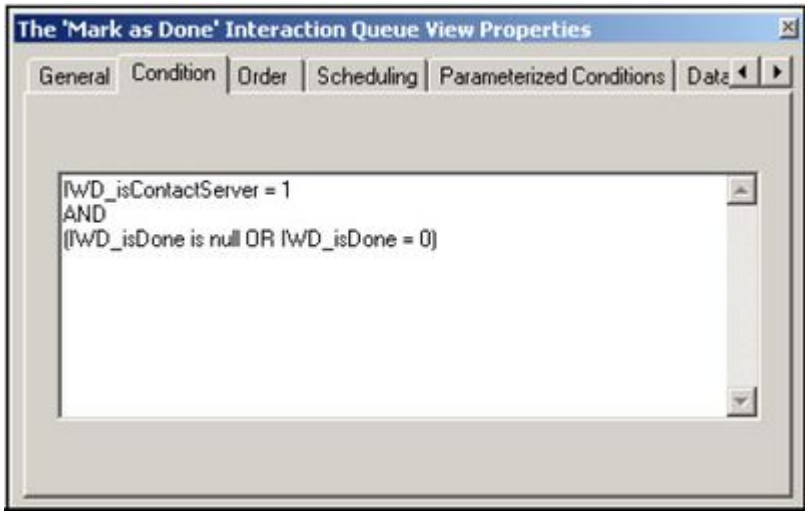
MarkInteractionAsDone Strategy

MarkInteractionAsDone Strategy

The purpose of this strategy is to update the Universal Contact Server (UCS) database to mark the interaction as done. This equates to setting the value in the `Status` column of the `Interactions` table to 3. UCS clients, such as Interaction Workspace, will then display the status of this interaction as done when the user looks at interactions they have previously processed.

Interactions have to satisfy the following conditions:

- The value of the attached data key `IWD_isContactServer` is `1`
- The value of the attached data key `IWD_isDone` is either null or `0` (zero)

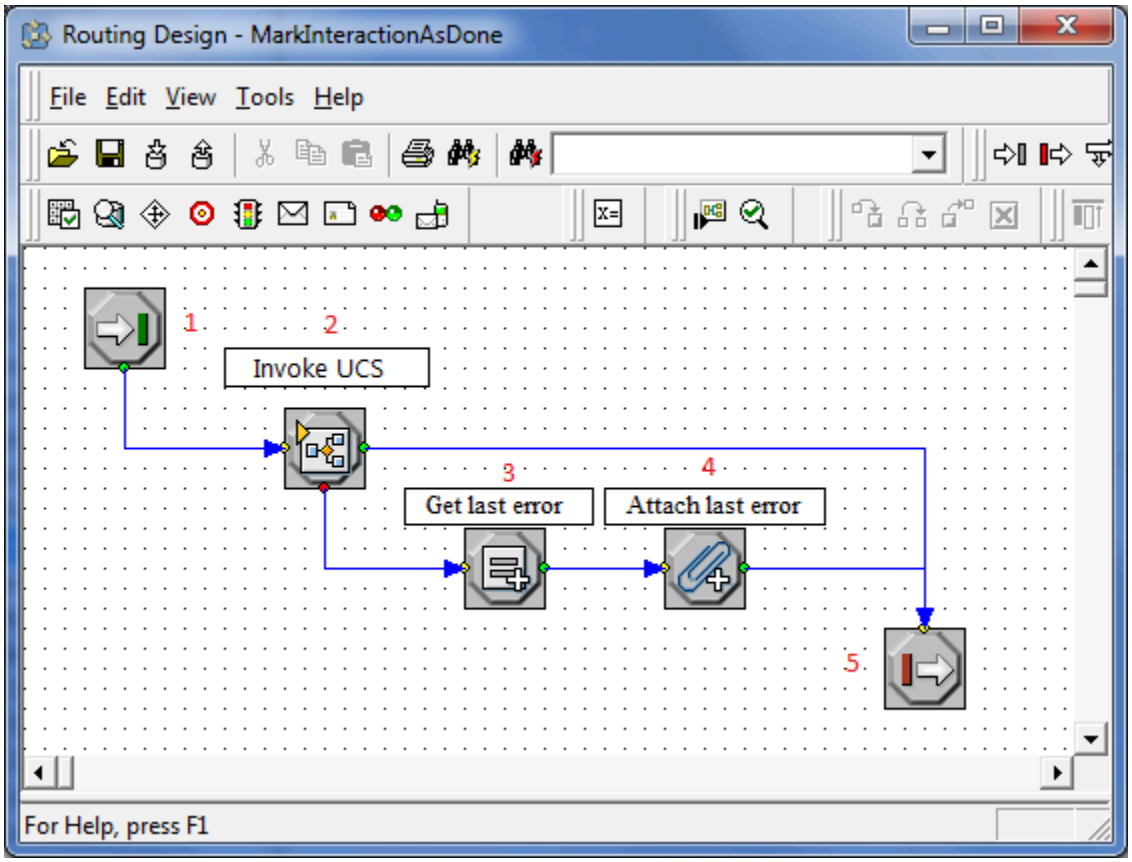


MarkInteractionAsDone View

This strategy processes interactions from the following queues:

- `iWD_Completed`
- `iWD_Canceled`
- `iWD_Rejected`

Flow Summary



Flow Detail

1. Entry to MarkInteractionAsDone strategy.
2. The InvokeUCS subroutine is invoked to complete interaction in the UCS database.
3. Reads `_last_error_str_` from InvokeUCS.
4. The last error is attached to user data as a key-value pair with the key `IWD_UCS_Error`.
5. Exit MarkInteractionAsDone strategy.

Removal Strategy

Removal Strategy

The purpose of this strategy is to delete expired interactions from the Interaction Server database.

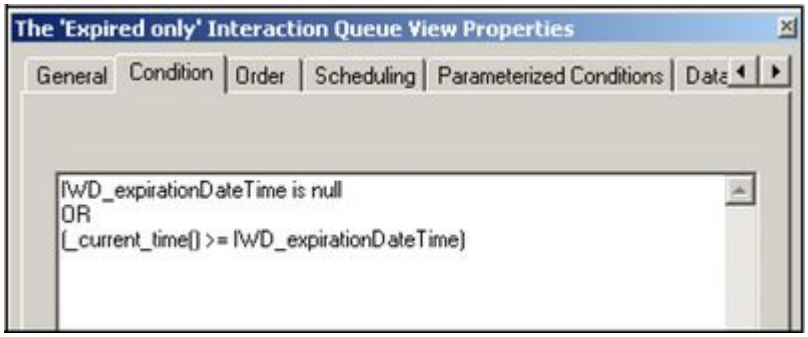
A key-value pair in user data with the key `IWD_expirationDateTime` contains information about when an interaction has to be `deleted`.

This strategy processes interactions from the following queues:

- `iWD_Completed`
- `iWD_Canceled`
- `iWD_Rejected`

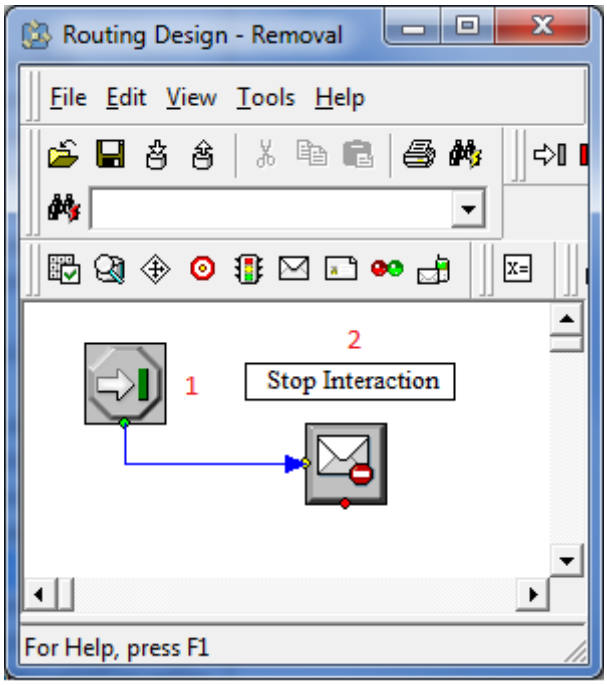
Interactions have to satisfy the following conditions:

- Interactions must either have the property `IWD_expirationDateTime` not set, or this property must have a time stamp which is in the past.
- Interactions are taken in the order they were submitted.



The 'Expired only' Interaction Queue View Properties

Flow Summary



Flow Detail

1. Start.
2. When interactions enter the Removal strategy, the processing of the interaction is stopped. This means that the interaction is deleted from the Interaction Server database.

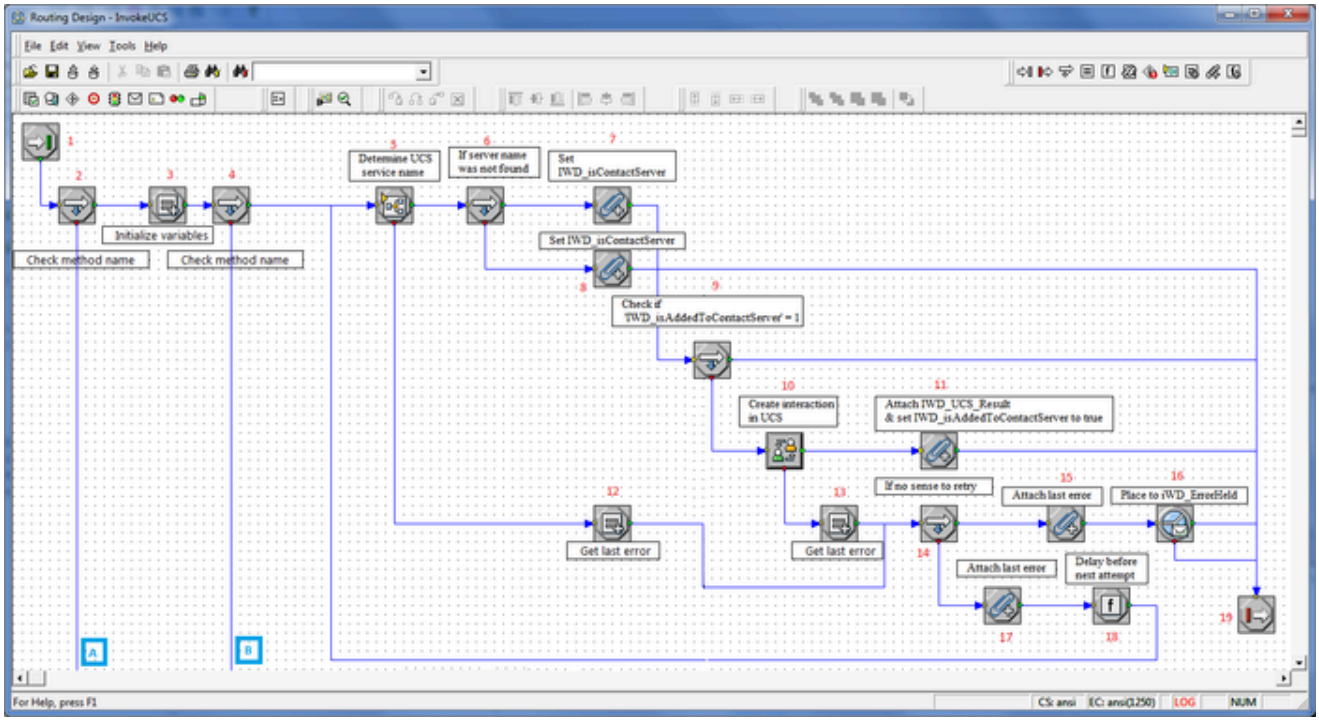
Invoke UCS Strategy

Invoke UCS Strategy

Flow Summary

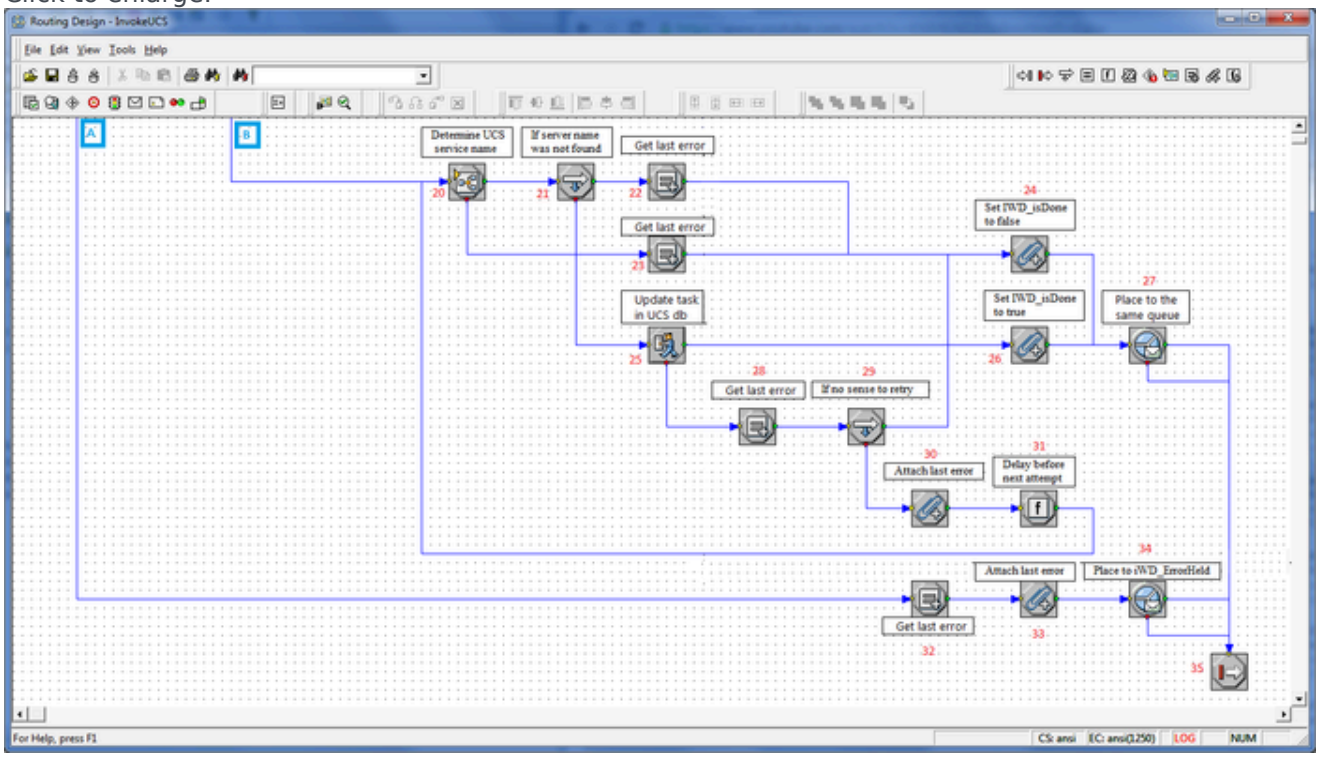
Part 1

Click to enlarge.



Part 2

Click to enlarge.



Flow Detail

1. Entry to InvokeUCS strategy.
2. Check if `in_method_name = 'Create' | in_method_name = 'OMInteractions'`.
3. Initialize variables:
   - `_tenant_id`—Read from task attribute `TenantId`.
   - `_interaction_id`—Read from task attribute `InteractionId`.
   - `delay_ms`—Specifies the delay (in milliseconds) between attempts to invoke UCS.
4. Check if `in_method_name = 'Create'`.
5. The DetermineESPServerName subroutine is invoked to determine the correct ESP server name to use. The subroutine uses the List Object list called `ContactServerList`. This subroutine also sets up cases when there is reason to retry to invoke the ESP server.
6. If the subroutine was successful, a check is done to ensure the existence of the ESP Server name that was returned by the subroutine.
7. The value of the user data key `IWD_isContactServer` is set to 1.
8. The value of the user data key `IWD_isContactServer` is set to `0`.
9. URS checks to see if the value of the user data key `IWD_isAddedToContactServer` is equal to 1, indicating that the task is already written into the interaction history in the UCS database.
10. A new interaction is created in the UCS database for this iWD task. If that function is successful, flow goes to 11. The user data key `IWD_isAddedToContactServer` is updated to 1 to indicate that the task was successfully added to the interaction history in UCS. The result returned from the ESP call to UCS is written to the variable `IWD_UCS_Result`.
11. The user data key `IWD_isAddedToContactServer` is updated to 1 to indicate that the task was successfully added to the interaction history in UCS. The result returned from the ESP call to UCS (from See A new interaction is created in the UCS database, for this iWD task. If that function is successful is written to the variable `IWD_UCS_Result`.
12. If the subroutine fails an error is extracted.
13. If the creation of the interaction in UCS was unsuccessful, an error is extracted from user data.
14. A check is done to see if the error code is related to the ESP server communication.
15. This error is attached to user data as a key-value pair with the key `IWD_UCS_Error`.
16. The interaction is placed in the `iWD_ErrorHeld` queue.
17. This error is attached to user data as a key-value pair with the key `IWD_UCS_Error`.
18. A delay is introduced, based on the value of the `_delay_ms` variable. The flow goes back to 5 to retry the connection to the ESP server.
19. Exit InvokeUCS strategy
20. The DetermineESPServerName subroutine is invoked to determine the correct ESP server name to use. The subroutine uses the List Object list called ContactServerList. This subroutine also sets up cases when there is reason to retry to invoke the ESP server.
21. If the subroutine was successful, a check is done to ensure the existence of the ESPserver name that was returned by the subroutine.

22. An error is extracted from user data.
23. An error is extracted from user data.
24. The value of the user data key `IWD_isDone` is set to 0.
25. The strategy calls a method on the Universal Contact Server to set the status of the interaction to 3, indicating that the interaction is done.
26. The value of the user data key `IWD_isDone` is set to 1.
27. The interaction is returned to its previous queue.
28. If the invocation of the method on the UCS fails, an error is extracted.
29. If it makes sense to retry updating the interaction record in UCS.
30. The last error code is attached to the interaction with the user data key `IWD_UCS_Error`.
31. A delay is introduced into the processing. Flow returns to step 20.
32. The last error is attached to user data as a key-value pair with the key `IWD_UCS_Error` when `in_method_name` is not set to `'Create'` or `in_method_name = 'OMInteractions'`.
33. The last error code is attached to the interaction with the user data key `IWD_UCS_Error`.
34. The interaction is placed in the `iWD_ErrorHeld` queue.
35. Exit InvokeUCS strategy.

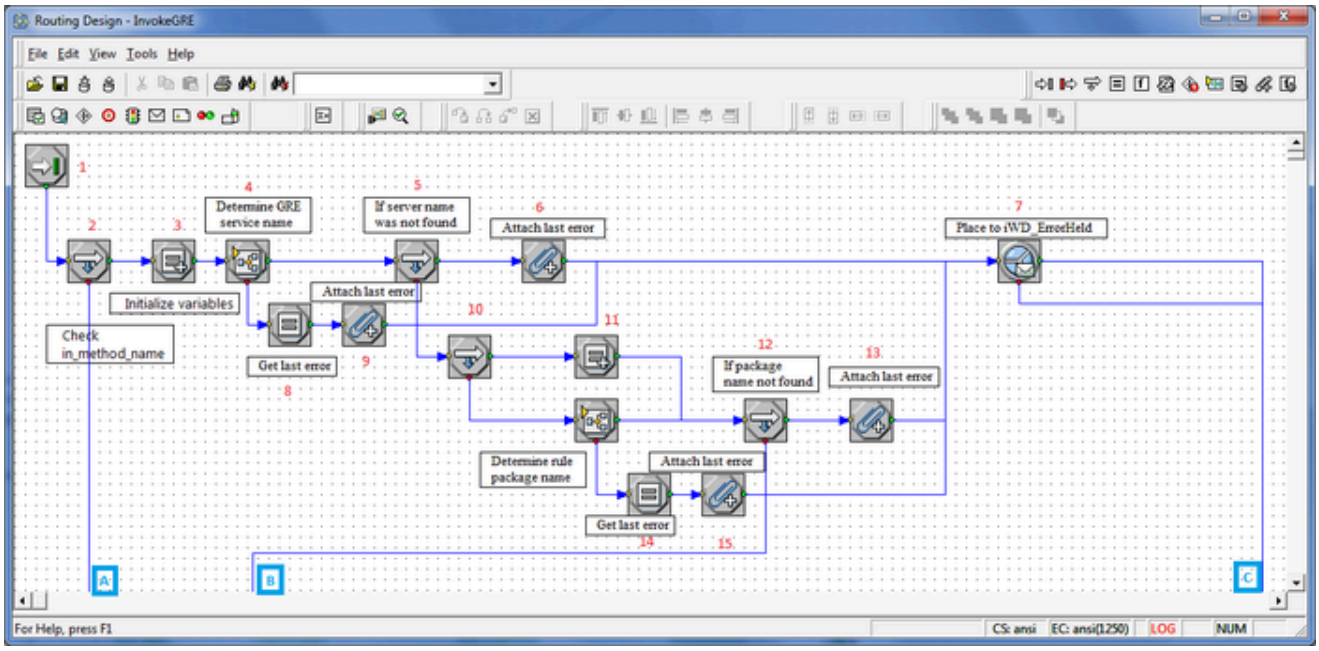
Invoke GRE Strategy

Invoke GRE Strategy

Flow Summary

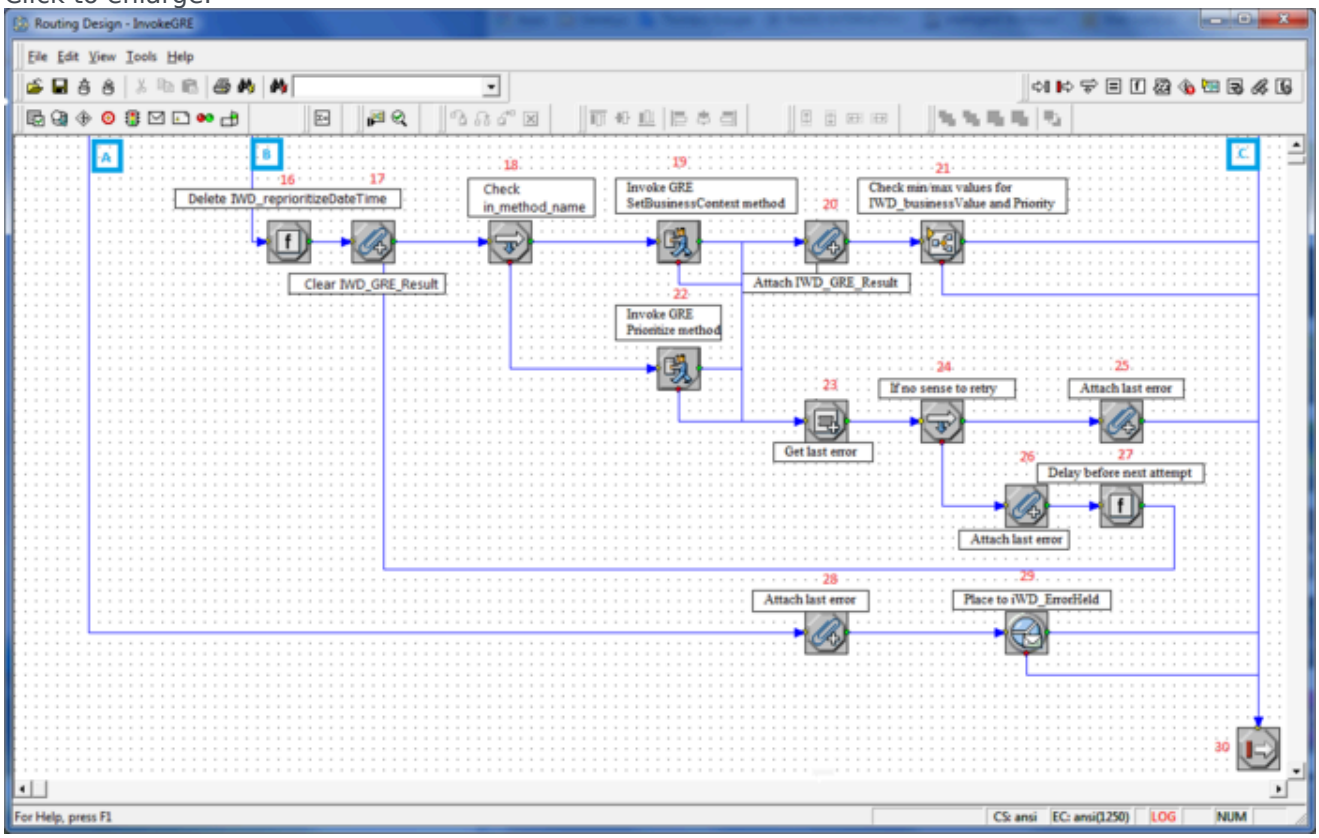
Part 1

Click to enlarge.



Part 2

Click to enlarge.



Flow Detail

1. Entry to InvokeGRE strategy.
2. Check if in_method_name is set to `SetBusinessContext` or `Prioritize`.
3. A variable is initialized—`_delay_ms` specifies the delay (in milliseconds) between attempts to invoke rules.
4. The DetermineESPServerName subroutine is invoked to determine the name of the Genesys Rules Engine Application. The subroutine uses the List Object list `GREServerList`.
5. If the subroutine was successful, a check is done to ensure the existence of the ESP server name that was returned by the subroutine. If the ESP server name was found, the flow goes to 10. The DetermineRulePackageName subroutine is invoked to determine the name of the rule package that the Genesys Rules Engine will be invoking to evaluate the classification rules.
6. If the ESP server name was not found, this error is attached to user data as a key-value pair with the key `IWD_GRE_Determination_Error`.
7. The interaction is placed in the `iWD_ErrorHeld` queue.
8. If the subroutine fails an error is extracted.
9. This error is attached to user data as a key-value pair with the key `IWD_GRE_Determination_Error`.
10. Check if `in_custom_package_name` was published to this subroutine. If it is set then `in_custom_package_name` will be run. Otherwise package name needs to be found in `Iwd_Package_List`.
11. Assign `in_custom_package_name` to `_gre_package_name` and set `_return_code` to `0`.

    11a) The DetermineRulePackageName subroutine is invoked to determine the name of the rule

    package that the Genesys Rules Engine will be invoking to evaluate the classification rules.

12. If the rule package name was found, the flow goes to Step 16.
13. If the rule package name was not found, this error is attached to user data as a key-value pair with the key `IWD_Rule_Package_Determination_Error`.
14. If the subroutine fails an error is extracted.
15. This error is attached to user data as a key-value pair with the key `IWD_Rule_Package_Determination_Error`.
16. Delete `IWD_reprioritizeDateTime` from attached data.
17. Delete `RulePhase` and `IWD_GRE_Result` before Invoke GRE.
18. Check if `in_method_name` = `SetBusinessContext`.
    - If `in_method_name` is set to `SetBusinessContext` then the process calls classification rules in GRE.
    - If `in_method_name` is set to `Prioritize` then the process calls prioritization rules in GRE.
19. An ESP request is sent to the Genesys Rules Engine to evaluate the classification rules.
20. The ESP result is attached to user data as a key-value pair with the key ---- IWD_GRE_Result.

21. CheckBusinessValueAndPriority subroutine is called to veriy if `IWD_businessValue` and `Priority` have correct values.
22. An ESP request is sent to the Genesys Rules Engine to evaluate the prioritization rules.
23. The last Interaction Server-related error is extracted from a variable.
24. A check is done to see if the error code is related to the ESP server communication.
25. The last error is attached to user data as a key-value pair with the key `IWD_GRE_Error`.
26. The last error is attached to user data as a key-value pair with the key `IWD_GRE_Error`. If not, the value of the `_counter` variable is incremented by 1.
27. A delay is introduced, based on the value of the `_delay_ms` variable. The flow goes back to 18 to retry the connection to the ESP server. The result from the ESP call to the Genesys Rules Engine is attached to the interaction as user data, with the key `IWD_GRE_Result`. This key-value pair will have the following format:

    ```
    return:ok| NumberOfRulesApplied:<number of applied rules>|
    RulesApplied:<rule 1 id> <rule1 name>, <rule2 id> <rule2
    name>, ...
    The following example shows what the result might look like:
    AttributeUserData [list, size (unpacked)=168] = 'ESP_Result'
    [str] =
    "return:ok|NumberOfRulesApplied:12|
    RulesApplied:McrSlt1GlbClsf1
    McrSlt1GlbClassification1, McrSlt1GlbClsf2
    McrSlt1GlbClassification2"
    ```

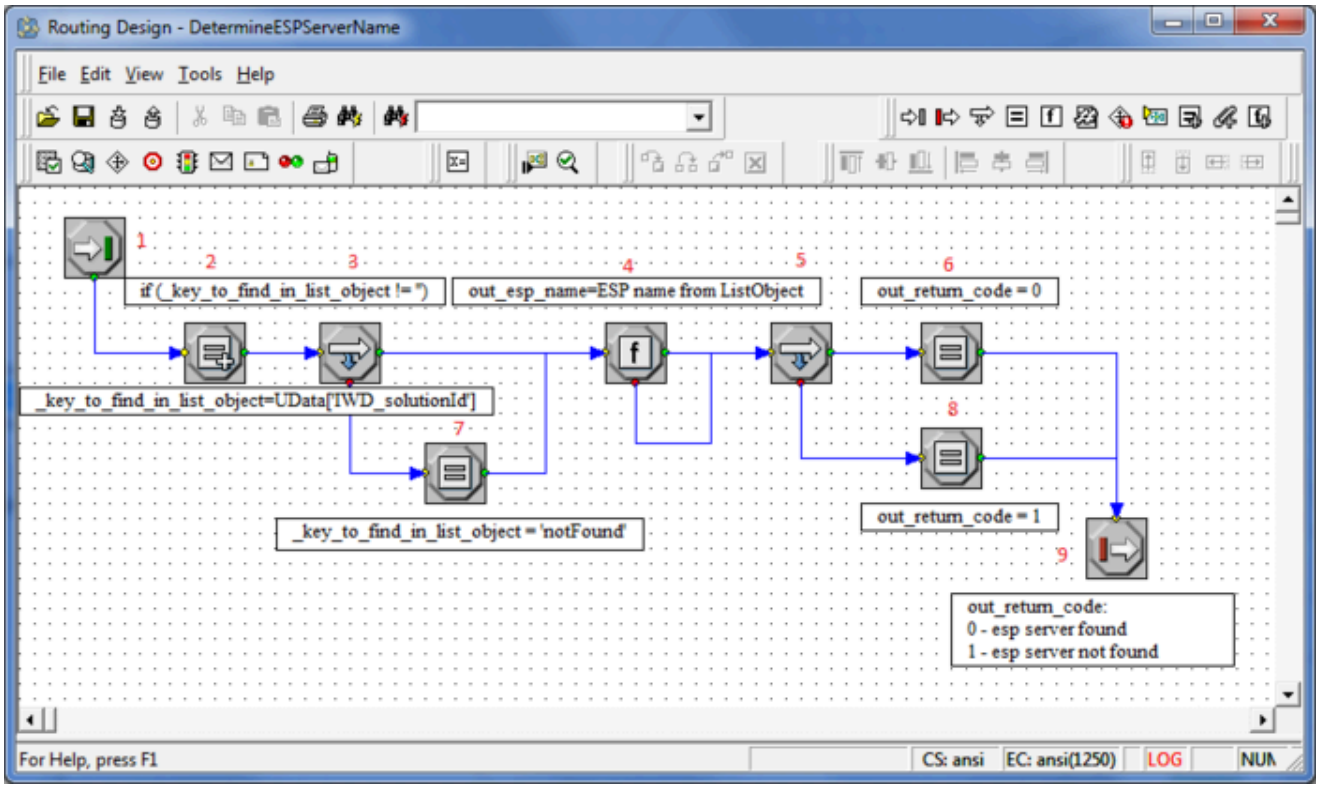
28. The last error is attached to user data as a key-value pair with the key `IWD_GRE_Error` when `in_method_name` is not set to `SetBusinessContext` or `Prioritize`.
29. The interaction is placed in the `iWD_ErrorHeld` queue.
30. Exit InvokeGRE subroutine.

DetermineESPServerName Subroutine

DetermineESPServerName

Flow Summary

Click to enlarge.



Flow Detail

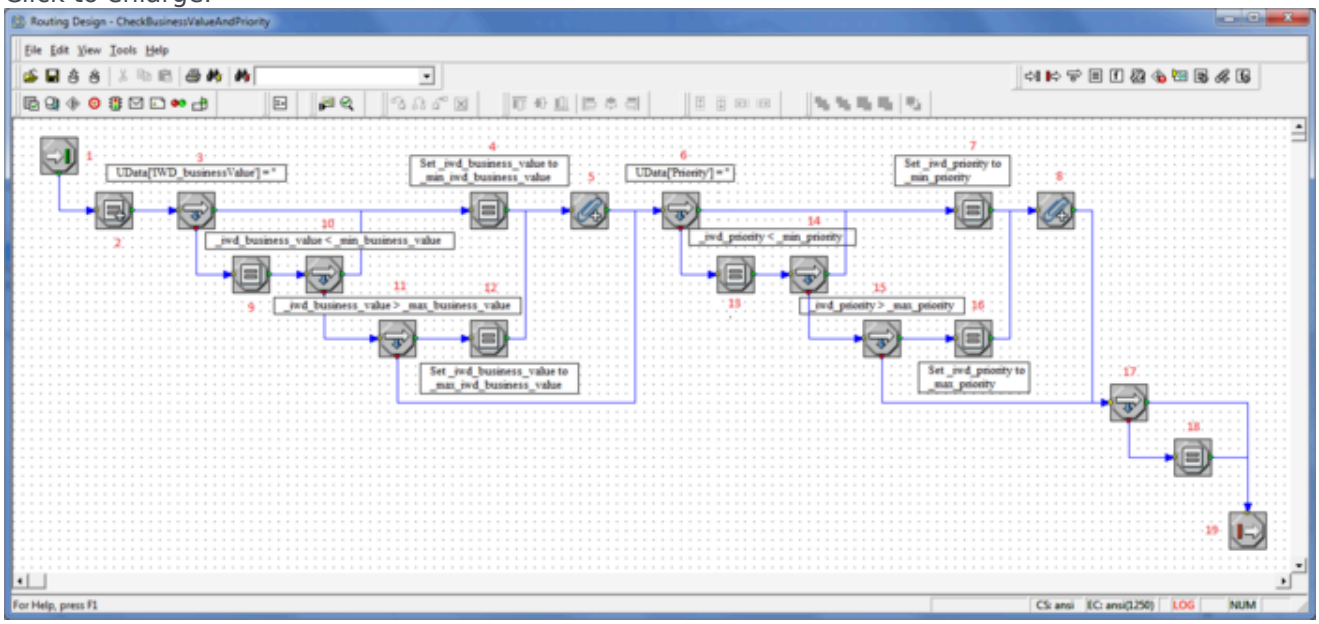
1. Entry to DetermineESPServerName subroutine.
2. Initialize variables:
   - `_key_to_find_in_list_object`—Assign task `IWD_SolutionId`.
   - `out_esp_name`—Set its value to empty string.
3. Check if `_key_to_find_in_list_object` is not empty.
4. Search `_key_to_find_in_list_object` in `Iwd_Esp_List`. Result will be assign to `out_esp_name`.
5. Check if `out_return_code` is not empty.
6. If `out_esp_name` is not empty then set `out_return_code` to `0`.
7. Assign `notFound` string to `_key_to_find_in_list_object`.
8. If `out_return_code` is empty then set `out_return_code` to 1.
9. Exit DetermineESPServerName subroutine.

CheckBusinessValueAndPriority Subroutine

CheckBusinessValueAndPriority

Flow Summary

Click to enlarge.



Flow Detail

1. Entry to the CheckBusinessValueAndPriority subroutine.
2. Variables are initialized:
   - `out_return_code`—Return code returned by this subroutine at the exit.
   - `_min_business_value`—Minimum available business value.
   - `_max_business_value`—Maximum available business value.
   - `_min_priority`—Minimum available priority.
   - `_max_priority`—Maximum available priority.
3. Check if task has set `IWD_businessValue` attribute.
4. If `IWD_businessValue` attribute was not set then it will be set to `_min_business_value`.
5. `IWD_businessValue` attribute is attached to the task.
6. Check if task has set `Priority` attribute.
7. If `Priority` attribute was not set then it will be set to `_min_priority`.

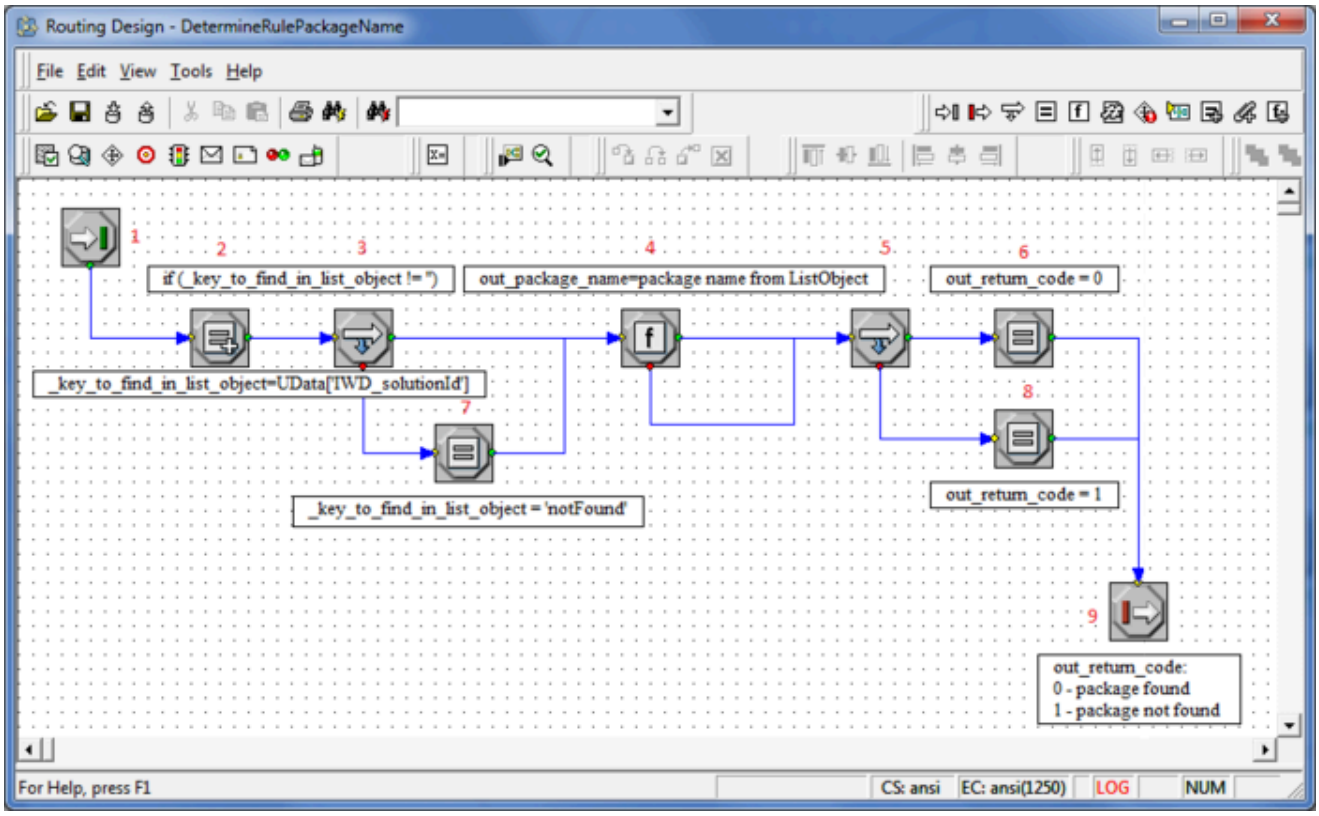
8. Priority attribute is attached to the task.
9. Read IWD_businessValue attribute from the task and put it in _iwd_business_value variable.
10. Check if _iwd_business_value is leass than _min_business_value. If true then flow goes to 4.
11. Check if _iwd_business_value is greater than _max_business_value.
12. If _iwd_business_value is greater than _max_business_value then set _iwd_business_value to _max_business_value.
13. Read Priority attribute from the task and put it in _iwd_priority variable.
14. Check if _iwd_priority is less than _min_priority. If true then flow goes to 7.
15. Check if _iwd_priority is geater than _max_priority.
16. If _iwd_priority is reater than _max_priority then set _iwd_priority to _max_priority.
17. Check if _iwd_business_value and _iwd_priority are in range of allowed values.
18. If _iwd_business_value and _iwd_priority are out of scope then out_return_code will be set to 1.
19. Exit CheckBusinessValueAndPriority subroutine.

DetermineRulePackageName Subroutine

DetermineRulePackageName

Flow Summary

Click to enlarge.



Flow Detail

1. Entry to DetermineRulePackageName subroutine.
2. Initialize variables:
   - `_key_to_find_in_list_object`—Assign task `IWD_SolutionId`.
   - `out_package_name`—Set its value to empty string.
3. Check if `_key_to_find_in_list_object` is not empty.
4. Search `_key_to_find_in_list_object` in `Iwd_Package_List`. Result will be assign to `out_package_name`.
5. Check if `out_return_code` is not empty.
6. If `out_package_name` is not empty then set `out_return_code` to `0`.
7. Assign `notFound` string to `_key_to_find_in_list_object`.
8. If `DetermineRulePackageName` is empty then set `out_return_code` to `1`.
9. Exit DetermineRulePackageName subroutine.