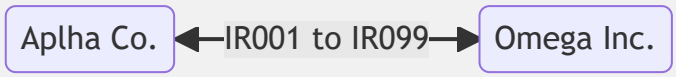
Multi-source eBonding *--DRAFT--*

About

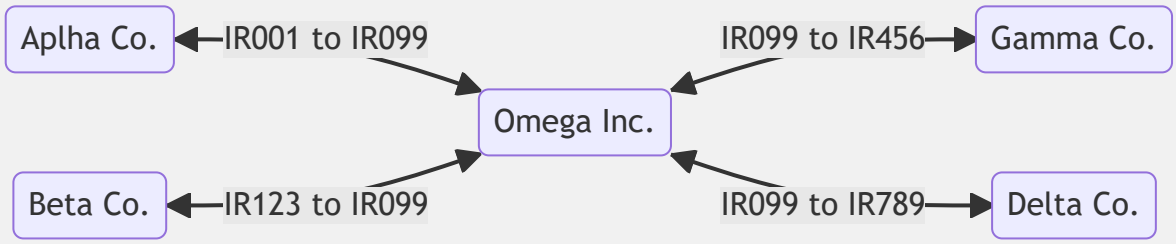
eBonding is a term used to describe the electronic passing of information for business-to-business (B2B) operations. Typically eBonding is used to keep two system of records in sync. A record on one system has the same information for a record on a different system. Each system passes information between each other as information on the record changes; hence both records contain the same information. Businesses setup and configure eBonding as a means to maintain a system of record they control automatically saving time and resources; i.e., billing, audit, automation, etc. etc..

Example: Alpha Co., is eBonded to Omega Inc.. Information between the two different companies' tickets are shared with each other digitally.



Multi-source eBonding is when you have three or more systems coordinating together on the same record from multiple sources. Multi-source eBonding is a mature B2B model for multi-supplier strategies, where one company coordinates with different multiple suppliers that provide various services within an organization. The advantage of a multi-supplier model is the model provides an organization the ability to right-size their enterprise needs across many suppliers that render a variety of services. Versus letting a single supplier render all operational needs within an organization.

Example: Alpha Co., Beta Co., Gamma Co., and Delta Co., are all eBonded to Omega Inc.. Information is shared across multiple tickets and additional logic is handled by Omega Inc.; determining how that information is shared across all suppliers.



This article is to help ServiceNow admins to setup multi-source eBonding framework for incident tickets. Incidents are normally a prime candidate to be shared across multiple suppliers. When an outage to a service is reported, there could be multiple suppliers working together to resolve the incident. Also if a supplier is utilized as an external call center, then routing incoming incident tickets to the appropriate secondary supplier needs to be performed. Given these two scenarios enterprises typically model, make incidents the first type of ticket to build a multi-source eBond framework. The same framework can be applied to other ticket types in ServiceNow with some minor adjustments.

Progressive thoughts

Here are the progressive thoughts as multi-source eBonding is being developed to go back and refer too.

- The enterprise will eBond with suppliers, so therefore is makes sense to create company records in the core_company table for each supplier.
- Company records need to be associated with the local ServiceNow accounts which are used by the supplier to eBond with ServiceNow. This will help you keep track what ServiceNow account is used by the supplier to make their RESTful API calls.
- eBond enabled companies only need to be associated with accounts that have the role of **eBond Account**.
- eBonding is triggered based on the **Assignment Group** a record to assigned to. Therefore, group (sys_user_group) records need to associated with the companies that ServiceNow are eBonded with.

Then in a business rule, if the assignment group company is eBonded, then continue to eBond operations.
- Turning off/on eBonding is as simple as toggling the company record in the core_company table.

Follow along

Each section will include an update set for those that wish to install the multi-source eBonding on their instance and the instructions to create the update set. ServiceNow offers many ways to solve a problem or configure an operational model. These instructions are a way and does not represent the way on creating a multi-source eBonding framework. The best way to use these instructions is to read through them and see how the framework can be adopted and modified to fit your organization.

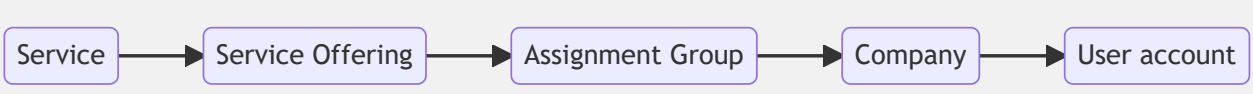
> **Update sets**
>
> The instructions do not cover how-to use update sets or the nuisances of update sets. You can explore on how to leverage update sets within the online ServiceNow documentation.

> **Terms & Meanings**
>
> - Supplier: The legal entity that ServiceNow connects too. You can substitute supplier for vendor, partner, or the like.
> - Company: A record from the core_company table in ServiceNow that represents the supplier.

> **Modifying OOTB tables**
>
> The article focuses on creating a robust framework tied closely to the out of the box (OOTB) tables already existing in ServiceNow. There are alternative implementation solutions within ServiceNow to accomplish the same result. Your mileage may vary depending on scope defined by and practices set by your organization. This framework is meant to be as open and flexible to meet varying modifications to suit present and future requirements.

Foundation setup

The foundation setup is based on the premise that when an incident is created the *Service* and *Service Offering* within an incident are mandatory fields. This is a best practice to follow with a multi-supplier strategy for the enterprise. The reason is this removes the question "who supports this" and focuses on the true issue of "what is service is not working?" Who solves the incident is not of any concern to most customers and requiring customers or fulfillers to know what assignment group to assign a ticket in large enterprise environments is unreasonable and prone to misrouted tickets.



The *Service* within an incident helps narrow the domain scope where the problem resides. The *Service Offering*, within an incident, should be limited to the service offerings with the *Service* and determines who the ticket is assigned to in the *Assignment group*. The *Assignment group* will be linked to a supplier (company record). This allows enterprises the ability to swap out suppliers across various services within ServiceNow quickly and easily reducing long term run and maintain costs.

> **Assignment groups**
>
> Normally an *Assignment group* will contain all the members that support the *Service Offering** within an incident. In the eBond case it is normal if their are no members of an *Assignment group*. It is also normal to have members in an *Assignment group* that is eBonded with a supplier as well; where the enterprise allows the supplier to log into the enterprise ServiceNow to fulfill tickets. Either scenario still works with eBonding.

The supplier will have a company record that is linked to a user record that represents the eBond account the supplier will use to make inbound RESTful calls. This will become a means to quickly identify what account a company uses for eBonding and creates a "lockout" point for security.

Demo data

This how-to provides demo data that you can use to test the multi-source eBonding framework. The scenario is your organization has hired Alpha Co., Beta Co., and Gamma Co. to supply IT services across your enterprise. Alpha Co. is your call-center where your customers call to request services and report incidents. Beta Co. supplies data center operations and Gamma Co. supplies backup and recovery operations for the data center. An incident has been reported to Alpha Co. at a data center where both Beta Co. and Gamma Co. need to work together to resolve the incident.

Update set

eBond Collegiality v1.0

Add eBond account role

The *eBond Account* role denotes which accounts are used by suppliers to eBond with your instance of ServiceNow. Each supplier should have their own local ServiceNow account that the supplier will use to connect to your instance of ServiceNow passing inbound REST payloads. This role will be used in various ways in the multi-source eBond framework.

1. Navigate to **User Administration** > **Roles**, click **New**.
2. Under *Role New Record*, fill in the following fields:
    - Name: eBond Account
    - Description: Denotes the sys_user account is to be used for eBonding activities.
3. Click **Submit**, which will create the new role.

Demo Data - User accounts

Setting up two accounts that will be used by two suppliers Alpha Co. and Beta Co. to make RESTful calls for eBonding.

1. Navigate to **User Administration** > **Users**, click **New**.
2. Under the **User New Record** section, fill in the following field:
    - User ID: eBondAlpha
    - First name: Alpha Co
    - Last name: eBond Account
3. In the record header, right-click and select **Save**.
4. In the **Roles** tab, click "Edit..."
5. Under the **Collection** list, find the *eBond Account* role and add it **>** to the **Roles List**.
6. Click **Save**.
7. Click **New**.
8. Under the **User New Record** section, fill in the following field:
    - User ID: eBondBeta
    - First name: Beta Co
    - Last name: eBond Account
9. In the record header, right-click and select **Save**.
10. In the **Roles** tab, click "Edit..."
11. Under the **Collection** list, find the *eBond Account* role and add it **>** to the **Roles List**.
12. Click **Save**.
13. Click **New**.
14. Under the **User New Record** section, fill in the following field:
    - User ID: eBondGamma
    - First name: Gamma Co
    - Last name: eBond Account
15. In the record header, right-click and select **Save**.
16. In the **Roles** tab, click "Edit..."
17. Under the **Collection** list, find the *eBond Account* role and add it **>** to the **Roles List**.
18. Click **Save**.

Changes to core_company table

We need two new columns in the core_company table; *eBond account* and *eBonded*. The *eBond account* is a reference to sys_user table that links which account the company uses to eBond with your instance. The *eBonded* is a true or false field that we can use to turn off eBonding at the company level.

1. Navigate to **System Definition** > **Tables** and open the *core_company* table.
2. Under the **Columns** tab, click **New** to create a new column.
3. Under the **Dictionary Entry New Record** section, fill in the following fields:
    - Type: Reference
    - Column label: eBond account
4. Under the **Reference Specification** tab, fill in the following fields:
    - Table to reference: User [sys_user]

    - Reference qual condition:
        - Active is true AND
        - Roles is eBond Account
5. Click **Submit**.
6. Under the **Columns** tab, click **New** to create a new column.
7. Under the **Dictionary Entry New Record** section, fill in the following fields:
    - Type: True/False
    - Column label: eBonded
8. Under the **Default value** tab, fill in the following field:
    - Default value: false
9. Click **Submit**.

OPTIONAL: Modify the view on company records to see the *eBond account* and *eBonded* fields.

1. Navigate to **Organization** > **Companies**, click on any of the company records.
2. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Company form**.
3. Under the **Available** list, find the *eBond account* and *eBonded* fields and add them **>** to the **Selected** list.
4. Move the *eBond account* and *eBonded* fields up the **Selected** list **^** to be below the *Stock price*.
5. Click **Save**.

OPTIONAL: Modify the view on company records to see the *eBond account* only if the *eBonded* field is true.

1. Navigate to **System UI** > **UI Polices**, click **New**.
2. Under the **UI Policy New record** section, fill in the following fields:
    - Table: core_company
    - Short description: Show eBond account in company
3. Under the **When to apply** tab, fill in the following fields:
    - Conditions:
        - eBonded is true
4. In the record header, right-click and select **Save**.
5. In the **UI Policy Actions**, click **New**.
6. Under the **UI Policy Action New record** section, fill in the following fields:
    - Field name: eBond account
    - Mandatory: True
    - Visible: True
7. Click **Submit**.
8. Click **Update**.

> **Demo Data - Company records**
>
> Setting up two company records for the suppliers Alpha Co. and Beta Co..
>
> 1. Navigate to **Organization** > **Companies**, click **New**.
> 2. Under the **Company New Record** section, fill in the following field:
>   - Name: Alpha Co.
>   - eBonded: True

    - eBond account: Alpha Co eBond Account
3. Click **Submit**.
4. Click **New**.
5. Under the **Company New Record** section, fill in the following field:
    - Name: Beta Co.
    - eBonded: True
    - eBond account: Beta Co eBond Account
6. Click **Submit**.

Associate groups with companies

FILL ME IN

1. Navigate to **System Definition** > **Tables** and open the *sys_usr_group* table.
2. Under the **Columns** tab, click **New** to create a new column.
3. Under the **Dictionary Entry New Record** section, fill in the following fields:
    - Type: Reference
    - Column label: Company
4. Under the **Reference Specification** tab, fill in the following fields:
    - Table to reference: core_company
5. Click **Submit**.
6. Click **Update**.

OPTIONAL: Modify the view on group records to see the *Company* field.

1. Navigate to **User Administration** > **Groups**, click on any of the group records.
2. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Group form**.
3. Under the **Available** list, find the *Company* field and add it **>** to the **Selected** list.
4. Move the *Company* field up the **Selected** list **^** to be below the *Name*.
5. Click **Save**.

> **Demo Data - Assignment Groups**
>
> Setting up two support groups; *Data Center Operations Support* and *Backup and Recovery Support*. The reason the group names are not associated with the names of the suppliers is a run and maintain operation decision. Most of the time a supplier will support multiple *Service offerings*, when an organization desires to replace a supplier, instead of updating multiple *Service offerings*, only the support group *Company* field need to be updated. This is a minor detail as many organizations choose to name the assignment groups after the supplier and that is perfectly fine.
>
> 1. Navigate to **User Administration** > **Groups**, click **New**.
> 2. Under the **Group New Record** section, fill in the following fields:
>   - Name: Data Center Support
>   - Company: Alpha Co.
> 3. Click **Submit**.
> 4. Click **New**.

5. Under the **Group New Record** section, fill in the following fields:
    - Name: Backup and Recovery Support
    - Company: Beta Co.
6. Click **Submit**.

Service, service offering, and assignment group alignment in incidents

Ticket mis-routing is problem in all services organization and surveys show that customer satisfaction reduces the more a ticket is re-routed. To reduce ticket mis-routes, the service, service offering, and assignment group need to be in alignment. It is unreasonable to expect that fulfillers know which support group supports what services. In large enterprises with hundreds of various services, internal teams, and external suppliers ticket mis-routing is common without alignment between all three fields.

> **Optional setup**
>
> This section is optional, but recommended for any ServiceNow environment; eBonded or not. For organizations that employ a multi-supplier strategy, this section is highly recommended.

1. Navigate to **System Definition** > **Dictionary**.
2. Under the list view search, fill in the the following search fields and hit enter:
    - Table: task
    - Column name: business_service
3. Open the *task* record.
4. Click on the **Reference Specification** tab, fill in the following field:
    - Reference qual condition:
        - Class is not Offering
5. Click **Update**.
6. Navigate to **System Definition** > **Business Rules**, click **New**.
7. Under the **Business Rule New Record** section, fill in the following fields:
    - Name: Incident Auto-assignment Group
    - Table: incident
    - Advanced: True
8. Under the **When to run** tab, fill in the following fields:
    - Insert: True
    - Update: True
    - Filter Conditions:
        - Service changes OR
        - Service offering changes
9. Under the **Advanced** tab, fill in the following fields:
    - Condition: gs.isInteractive()
    - Script:

```
(function executeRule(current, previous /*null when async*/ ) {
```

```
     if (!gs.nil(current.service_offering.assignment_group))
         current.assignment_group = current.service_offering.assignment_gro
     else
         current.assignment_group = current.business_service.assignment_gro
})(current, previous);
```

10. Click **Submit**.

Demo Data - Service and Service Offerings

Setting up two services and three service offerings for Alpha Co., Beta Co., and Gamma Co.

Service	ServiceOffering	Support Group
Service Management	Call Center Operations	Call Center Support
IT Infrastructure	Data Center Operations	Data Center Support
IT Infrastructure	Backup & Recovery	Backup and Recovery Support

1. Navigate to **Configuration** > **Services**, click **New**.
2. Under the **Service New Record** section, fill in the following fields:
    - Name: Service Management
    - Service classification: Business Service
3. In the record header, right-click and select **Save**.
4. Under the **Offerings** tab, click **New**.
5. Under the **Offering New Record** section, fill in the following fields:
    - Name: Call Center Operations
    - Support Group: Call Center Support
6. Click **Submit**.
7. Click **Update**.
8. Click **New**.
9. Under the **Service New Record** section, fill in the following fields:
    - Name: IT Infrastructure
    - Service classification: Technical Service
10. In the record header, right-click and select **Save**.
11. Under the **Offerings** tab, click **New**.
12. Under the **Offering New Record** section, fill in the following fields:
    - Name: Data Center Operations
    - Support Group: Data Center Support
13. Click **Submit**.
14. Under the **Offerings** tab, click **New**.
15. Under the **Offering New Record** section, fill in the following fields:
    - Name: Backup & Recovery
    - Support Group: Backup and Recovery Support
16. Click **Submit**.
17. Click **Update**.

Associate incidents with eBond companies

Fulfillers will need the ability to eBond incidents with suppliers. And we have partially started the setup that a trigger point will be the *Assignment group* in an incident. Now we will start to add custom fields to the *incident* table to enhance further functionality that will support and aid in multi-source eBonding.

> **Assignment group**
>
> Later on we will configure how changing the *Assignment group* will affect the *eBonded with* field.

There are two main ways to eBond an incident to a supplier; first via the *Assignment group* and the second via a custom field called *eBonded with*. The *eBonded with* will serve a dual purpose. First, it will represent all the suppliers the ticket is eBonded with in a list. A fulfiller will then have the ability to quickly see what suppliers the ticket is eBonded too. Second, it will toggle eBonding and deBonding operations when suppliers are added or removed from the list. That way a fulfiller can initiate an eBond or deBond with a supplier without having to change the *Assignment group."

> **eBonded with**
>
> This field expects the fulfiller to know which suppliers to eBond with and the service offerings the supplier provides. If the fulfiller does not know the proper supplier, then they should use the *Service* and *Service offering* fields to set the *Assignment group* field to the appropriate group.

> **deBonding**
>
> deBonding is the opposite of eBonding by breaking the connection between the tickets. After a ticket has been deBonded from a company, the ticket can be re-eBonded to the company by adding the company back to the list. In the event of a re-eBonding, there is no guarantee the company ticket will be the same .

1. Navigate to **System Definition** > **Tables** and search for the *incident* table under the **Name** field.
2. Open the *Incident* record and click **New** under the **Columns** tab.
3. Under the **Dictionary Entry New Record** section, fill in the following fields:
    - Type: List
    - Column label: eBonded with
    - Column name: (this should default to u_eBonded_with)
4. Under the **Reference Specification** tab, fill in the following fields:
    - Reference: Company [core_company]
    - Reference qual condition:
        - eBonded is true AND
        - eBond Account is not empty
5. Click **Submit**.
6. Navigate to **Incident** > **All** and open any incident record.

7. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Company form**.
8. Under the **Available** list, find the *eBonded with* field and add it **>** to the **Selected** list.
9. Move the *eBonded with* field up the **Selected** list **^** to be below the *Assigned to*.
10. Click **Save**.

Custom tables

eBonding will require the creation of custom tables or the extension of existing tables. ServiceNow sets a limit on the number of custom tables an instance can have before additional costs are accrued/incurred. This how-to will create custom tables as there are no similar tables in ServiceNow that are proper candidates to extend. You however can take any of the base tables in ServiceNow and extend them adding the custom fields and it will work the same.

eBond relationships

The *u_eBond_relationship* table is a multi-relational table that represents a one to many record mapping for an incident ticket to be mapped to multiple supplier tickets.

The *u_eBond_relationship* table will contain the following custom fields:

Field	Description
Source	The ServiceNow record sys_id on the instance.
Source table	The table the ServiceNow record resides on.
Status	The status of the eBond.
State	The state of the eBond.
In	Inbound communication status from supplier.
Out	Outbound communication status to supplier.
Correlation Number	The supplier's ticket number or designation
Correlation ID	The supplier's ticket sys id or reference
Company	The ServiceNow supplier company record on the instance.
URL	Link to the supplier ticket.
Reflect	Changes made by the supplier are reflected in the ticket.

The fields relate the ServiceNow ticket to the various suppliers the ticket is eBonded too.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond Relationship
    - Name: u_ebond_relationship
    - New menu name: eBond

4. In the record header, right-click and select **Save**.
5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Table Name
    - Column label: Source table
    - Column name: (this should default to u_source_table)
7. Click **Submit**.
8. In the **Columns** table, click **New**.
9. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Document ID
    - Column label: Source
    - Column name: (this should default to u_source)
10. Under **Related Links**, click **Advanced view**.
11. Under the **Dependent Field** tab, click **Use dependent field**.
12. In the **Dependent on field**, select **Source table**.
13. Click **Update**.
14. In the **Columns** table, click **New**.
15. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Status
    - Column name: (this should default to u_status)
16. In the **Choice List Specification** tab, select *Dropdown with --None--*.
17. In the record header, right-click and select **Save**.
18. In the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	eBonded	eBonded
200	deBonded	deBonded

19. Click *Update**.
20. In the **Columns** table, click **New**.
21. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: State
    - Column name: (this should default to u_state)
22. In the **Choice List Specification** tab, select *Dropdown with --None--*.
23. In the record header, right-click and select **Save**.
24. In the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value

Sequence	Label	Value
100	New	1
200	In Progress	2
300	On Hold	3
400	Resolved	6
500	Closed	7
600	Canceled	8

25. Click *Update**.
26. In the **Columns** table, click **New**.
27. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: In
    - Column name: (this should default to u_in)
28. In the **Choice List Specification** tab, select *Dropdown with --None--*.
29. In the record header, right-click and select **Save**.
30. In the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	Up	up
200	Down	down

31. Click *Update**.
32. In the **Columns** table, click **New**.
33. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Out
    - Column name: (this should default to u_out)
34. In the **Choice List Specification** tab, select *Dropdown with --None--*.
35. In the record header, right-click and select **Save**.
36. In the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	Up	up
200	Down	down

37. Click *Update**.
38. In the **Columns** table, click **New**.

39. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Correlation Number
    - Column name: (this should default to u_correlation_number)
    - Max length: 40
40. Click **Submit**.
41. In the **Columns** table, click **New**.
42. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Correlation ID
    - Column name: (this should default to u_correlation_id)
    - Max length: 32
43. Click **Submit**.
44. In the **Columns** table, click **New**.
45. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Reference
    - Column label: Company
    - Column name: (this should default to u_company)
46. In the **Reference Specification** tab, fill in the following field:
    - Reference: Company [core_company]
    - Reference qual condition:
        - eBonded is true AND
        - eBond Account is not empty
47. Click **Submit**.
48. In the **Columns** table, click **New**.
49. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: URL
    - Column label: URL
    - Column name: (this should default to u_url)
    - Max length: 4000
50. Click **Submit**.
51. In the **Columns** table, click **New**.
52. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: True/False
    - Column label: Reflect
    - Column name: (this should default to u_reflect)
53. Under the **Default Value** tab, , fill in the following field:
    - Default value: false
54. Click **Submit**.
55. Click **Update**.

Relationship Support Script

Helper script to set the **In** and **Out** fields for the **u_ebond_relationship** table.

1. Navigate to **System Definition** > **Script Includes**, click **New**.
2. In the **Script Include New record**section, fill in the following fields:

- Name: eBondRelationship
- API Name: (this should default to global.eBondRelationship)
- Script:

```
var eBondRelationship = Class.create();
eBondRelationship.prototype = {
    initialize: function() {

    },

    // changes the state of the In field
    updateIn: function(relationship, value) {
        var gr = new GlideRecord('u_ebond_relationship');
        gr.addQuery('sys_id',relationship);
        gr.query();
        if (gr.next()) {
            gr.u_in = value;
            gr.update();
        }
        return;
    },

    // changes the state of the Out field
    updateOut: function(relationship, value) {
        var gr = new GlideRecord('u_ebond_relationship');
        gr.addQuery('sys_id',relationship);
        gr.query();
        if (gr.next()) {
            gr.u_out = value;
            gr.update();
        }
        return;
    },

    type: 'eBondRelationship'
};
```

3. Click **Submit**.

Stylize Relationship In/Out

Adding color to the **In** and **Out** fields for the **u_ebond_relationship** table. This is a quick visual aid in identifying the inbound and outbound state of the incident.

1. Navigate to **System UI** > **Field Styles**, click **New**.
2. In the **Style New record** section, fill in the following fields:
    - Table: eBond Relationship [u_ebond_relationship]
    - Field name: In
    - Value: Up
    - Style: background-color:green;color:green
3. Click **Submit**.
4. Click **New**.

5. In the **Style New record** section, fill in the following fields:
    - Table: u_ebond_relationship
    - Field name: In
    - Value: Down
    - Style: background-color:tomato;color:tomato
6. Click **Submit**.
7. Click **New**.
8. In the **Style New record** section, fill in the following fields:
    - Table: u_ebond_relationship
    - Field name: Out
    - Value: Up
    - Style: background-color:green;color:green
9. Click **Submit**.
10. Click **New**.
11. In the **Style New record** section, fill in the following fields:
    - Table: u_ebond_relationship
    - Field name: Out
    - Value: Down
    - Style: background-color:tomato;color:tomato
12. Click **Submit**.

Relating eBonded tickets

Associating the supplier tickets within the incident requires the creation of a relationship between the **incident** and **u_ebond_relationship** tables.

1. Navigate to **System Definition** > **Relationships**, click **New**.
2. In the **Relationship New record** section, fill in the following fields:
    - Name: eBond Relationships
    - Applies to table: Global [global]
    - Queries from table: eBond Relationship [u_ebond_relationship]
    - Query with:

```
current.addQuery('u_source_table', parent.getTableName());
current.addQuery('u_source', parent.sys_id);
```

3. Click **Submit**.
4. Navigate to **Incident** > **All**, click on any of the incident records.
5. Navigate to **Additional actions** > **Configure** > **Related Lists**, which will bring up the **Configuring related lists on Incident form**.
6. Under the **Available** list, find the *eBond Relationships* relationship and add it **>** to the **Selected** list.
7. Click **Save**.

---

eBond Properties

The *u_ebond_registry* table is similar to that of the *sys_properties* table in that it stores generic properties and values for eBonding. If you choose to use the *sys_properties* table, then make sure to look through the various scripts that reference the *u_ebond_registry* table in this how-to and change as appropriate.

The *u_ebond_registry* table will contain the following custom fields:

Field	Description
Supplier	"All" or the name of the supplier.
Key	The key variable name.
Value	The property variable value.
Note	Misc. information pertaining to the key/value pair.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond Registry
    - Name: u_ebond_registry
    - Add module to menu: eBond
4. In the record header, right-click and select **Save**.
5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Supplier
    - Column name: (this should default to u_supplier)
    - Max length: 40
7. Under the **Default Value** tab, fill in the following field:
    - Default value: All
8. Click **Submit**.
9. In the **Columns** table, click **New**.
10. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Key
    - Column name: (this should default to u_key)
    - Max length: 4000
11. Click **Submit**.
12. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Value
    - Column name: (this should default to u_value)
    - Max length: 4000
13. Click **Submit**.
14. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Note

- Column name: (this should default to u_note)
- Max length: 4000

15. Click **Submit**.
16. Click **Update**.

---

eBond Logs

The *u_ebond_log* table is similar to that of the *syslog* table in that it stores generic log information. If you choose to use the *syslog* table, then make sure to look through the various scripts that reference the *u_ebond_log* table. The *syslog* table has a granularity of seconds and ServiceNow performs multiple operations per second. Hence the order of the logs messages become out of order. It is recommended that you use the *u_ebond_log* to help with debugging as the *Index* column is a sequential integer to aid in showing the order of operations.

> **Evaluator messages**

The *u_ebond_log* table does not capture *Evaluator* messages from faulted JavaScript execution. Use the *syslog* table to check for failed JavaScript executions.

eBond Log Index Script Include

The *eBondLog* script include is a utility function that will increment the default value in the *index* field in the *[u_ebond_log]* table. The *index* value is based on the greatest index value in the table. If no rows exist, the *index* value is reset to zero.

1. Navigate to **System Definition** > **Script Includes**.
2. Click **New**.
3. Under the **Script Include New record** section, fill in the following fields:
   - Name: eBondLog
   - API Name: (this should default to global.eBondLog)
   - Script:

```
var eBondLog = Class.create();
eBondLog.prototype = {
    initialize: function() {
        u_direction = 'not set';
        u_location = 'not set';
        u_name = 'not set';
        u_source = 'not set';
        u_supplier = 'not set';
        u_message = 'not set';
        u_level = 'info';

        logLevel = -1;
        var reg = new GlideRecord('u_ebond_registry');
        reg.addQuery('u_supplier','All');
        reg.addQuery('u_key','inbound.log.level');
        reg.query();
        if (reg.next()) {
            logLevel = parseInt(reg.value);
```

```
        }
    },

    // returns the next default index value
    increment: function() {
        var gr = new GlideRecord('u_ebond_log');
        gr.orderByDesc('u_index');
        gr.setLimit(1);
        gr.query();
        var index = 0;
        if (gr.next()) {
            index = gr.u_index + 1;
        }
        return index; // return the calculated value
    },

    // records the log message
    insert: function () {
        // only log levels at or below what has been registered
        if (this.logLevel < parseInt(u_level)) {
            return;
        }

        var log = new GlideRecord('u_ebond_log');
        log.u_direction = this.u_direction;
        log.u_location = this.u_location;
        log.u_name = this.u_name;
        log.u_source = this.u_source;
        log.u_supplier = this.u_supplier;
        log.u_message = this.u_message;
        log.u_level = this.u_level;
        log.insert();
        return;
    },

    // records the log message
    write: function (level, message) {
        // only log levels at or below what has been registered
        if (this.logLevel < parseInt(u_level)) {
            return;
        }

        var log = new GlideRecord('u_ebond_log');
        log.u_direction = this.u_direction;
        log.u_location = this.u_location;
        log.u_name = this.u_name;
        log.u_source = this.u_source;
        log.u_supplier = this.u_supplier;
        log.u_message = message;
        log.u_level = level;
        log.insert();
        return;
    },

    type: 'eBondLog'
};
```

4. Click **Submit**.

eBond Log Index Dynamic Filter Options

The dynamic filter option *eBond Log Index Increment* is used in the default value in the *index* field in the *[u_ebond_log]* table as a reference to the script include *eBondLog*.

1. Navigate to **System Definition** > **Dynamic Filter Options**.
2. Click **New**.
3. Under the **Dynamic Filter Options New record** section, fill in the following fields:
    - Label: eBond Log Index Increment
    - Script: new eBondLog().increment();
    - Available for default: true
4. Click **Submit**.

The *u_ebond_log* table will contain the following custom fields:

Field	Description
Index	Auto-incrementing numeric value.
Direction	Choice value of *Inbound* or *Outbound*.
Supplier	Name of the supplier.
Location	Name of the ServiceNow component, i.e., script include, business rule, workflow, etc. etc..
Name	Label name of the record.
Source	Where within the record.
Message	The log message.
Level	Choice value of *High*, *Medium*, *Low*, or *Info*.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond Log
    - Name: u_ebond_log
    - Add module to menu: eBond
4. In the record header, right-click and select **Save**.
5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Direction
    - Column name: (this should default to u_direction)
7. In the record header, right-click and select **Save**.
8. Under the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	Inbound	inbound
200	Outbound	outbound

9. Click **Update**.
10. In the **Columns** table, click **New**.
11. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: Index
    - Column name: (this should default to u_index)
    - Read only: True
12. Under **Related Links**, click **Advanced view**.
13. Under the **Default Value** tab, fill in the following field:
    - Use dynamic default: true
    - Dynamic Default value: eBond Log Index Increment
14. Click **Submit**.
15. In the **Columns** table, click **New**.
16. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Supplier
    - Column name: (this should default to u_supplier)
    - Max length: 40
17. Click **Submit**.
18. In the **Columns** table, click **New**.
19. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Location
    - Column name: (this should default to u_location)
    - Max length: 40
20. Click **Submit**.
21. In the **Columns** table, click **New**.
22. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Name
    - Column name: (this should default to u_name)
    - Max length: 40
23. Click **Submit**.
24. In the **Columns** table, click **New**.
25. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Source
    - Column name: (this should default to u_source)
    - Max length: 80
26. Click **Submit**.
27. In the **Columns** table, click **New**.

28. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Message
    - Column name: (this should default to u_message)
    - Max length: 4000
29. Click **Submit**.
30. In the **Columns** table, click **New**.
31. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Level
    - Column name: (this should default to u_level)
32. In the record header, right-click and select **Save**.
33. Under the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	High	100
200	Medium	200
300	Low	300
400	Info	400

34. Click **Update**.
35. Click **Update**.

Create the *[u_ebond_registry]* record used to determine the depth of logs levels to record.

1. Navigate to **eBond** > **eBond Registries**.
2. Click **New**.
3. Under the **eBond Registry New record** section, fill in the following fields:
    - Supplier: All
    - Key: inbound.log.level
    - Value: 500
4. Click **Submit**.

Create the

eBond Data Map

The *u_ebond_data_map* table is a translation data table for inbound and outbound communication with suppliers.

The *u_ebond_data_map* table will contain the following custom fields:

Field	Description
Module	*All* or the name of the module.

Field	Description
Classification	Name of the classification.
Direction	*Inbound*, *Outbound*, or *Duplex*.
Supplier	*All* or the name of the supplier.
Source Value	The internal value for the instance.
Supplier Value	The supplier's value.
Note	Freeform to document various notes pertaining to the data map record.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond Data Map
    - Name: u_ebond_data_map
    - Add module to menu: eBond
4. In the record header, right-click and select **Save**.
5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Module
    - Column name: (this should default to u_module)
    - Max length: 40
7. Click **Submit**.
8. In the **Columns** table, click **New**.
9. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Classification
    - Column name: (this should default to u_classification)
    - Max length: 40
10. Click **Submit**.
11. In the **Columns** table, click **New**.
12. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Direction
    - Column name: (this should default to u_direction)
    - Max length: 40
13. In the record header, right-click and select **Save**.
14. Under the **Choices** tab, add the following records performing these steps:
    1. Click **New**.
    2. In the **Choice New record** section, fill in the field values listed below.
    3. Click **Submit**.

Sequence	Label	Value
100	Inbound	inbound

Sequence	Label	Value
200	Outbound	outbound
300	Duplex	duplex

15. Click **Update**.
16. In the **Columns** table, click **New**.
17. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Supplier
    - Column name: (this should default to u_supplier)
    - Max length: 40
18. Click **Submit**.
19. In the **Columns** table, click **New**.
20. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Source value
    - Column name: (this should default to u_source_value)
    - Max length: 4000
21. Click **Submit**.
22. In the **Columns** table, click **New**.
23. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Supplier value
    - Column name: (this should default to u_supplier)
    - Max length: 4000
24. Click **Submit**.
25. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Note
    - Column name: (this should default to u_note)
    - Max length: 4000
26. Click **Submit**.
27. Click **Update**.

---

eBond Data Map Script Include

Support script used in eBonding operations

1. Navigate to **System Definition** > **Script Includes**, click **New**.
2. In the **Script Include New record** section, fill in the following fields:
    - Name: eBondDataMap
    - API Name: (this should default to global.eBondDataMap)
    - Script:

```
var eBondDataMap = Class.create();
eBondDataMap.prototype = {
    // func: initialize
```

```
    // desc: initializes the JavaScript object
    // parm: n/a
    // retn: true or false
    initialize: function() {
        // static
        this.eLog = new eBondLog();
        this.eLog.u_direction = 'outbound';
        this.eLog.u_location = 'Script Includes';
        this.eLog.u_name = 'eBondDataMap';
        this.eLog.u_source = 'initialize';
        this.eLog.u_supplier = 'Unknown';
        this.eLog.write('Info', 'Entering.');

        this.eLog.write('Info', 'Exiting.');
    },

    // func: getSupplierValue
    // desc: retrieves the supplier's value from the u_ebond_mapping table.
    // parm: supplier - supplier name
    //       module - module value to query against
    //       classification - classification value to query against
    //       source_value - source value to query against
    //       default_value - if not found return this supplier value
    // retn: supplier value.
    getSupplierValue: function(supplier, direction, module, classification, sour
        this.eLog.u_source = 'getSupplierValue';
        this.eLog.u_supplier = supplier;
        this.eLog.write('Info', 'Entering.');

        var supplierValue = new GlideRecord("u_ebond_data_map");
        supplierValue.addQuery("u_supplier", supplier);
        supplierValue.addQuery("u_direction", direction);
        supplierValue.addQuery("u_module", module);
        supplierValue.addQuery("u_classification", classification);
        supplierValue.addQuery("u_source_value", source_value);
        supplierValue.query();

        if (supplierValue.next()) {
            this.eLog.write('Info', 'Exiting. Return: ' + supplierValue.u_suppli
            return supplierValue.u_supplier_value.toString();
        } else {
            this.eLog.write('Info', 'Exiting. Return: ' + default_value);
            return default_value;
        }
    },

    type: 'eBondDataMap'
};
```

3. Click **Submit**.

eBond REST Payloads

The *u_ebond_rest_payloads* table records all outbound REST API calls and their statuses. This table is also used to retry REST calls in the event of a failure; see **eBond Scheduled Job**.

eBond REST Payloads

The dynamic filter option *eBond REST Payload Index Increment* is used in the default value in the *index* field in the *[u_ebond_rest_payload]* table as a reference to the script include *eBondRestPayload*.

1. Navigate to **System Definition** > **Dynamic Filter Options**.
2. Click **New**.
3. Under the **Dynamic Filter Options New record** section, fill in the following fields:
    - Label: eBond REST Payload Index Increment
    - Script: new eBondLog().increment();
    - Available for default: true
4. Click **Submit**.

The *u_ebond_rest_payloads* table will contain the following custom fields:

Field	Description
Supplier	The name of the supplier.
Retry count	The number of retries performed executing the RESTful call to the supplier.
Retry	True or False.
REST Message	ServiceNow REST message label.
Payload	REST data payload for the supplier.
Format	The format of the payload; e.g., JSON, XML, SOAP, etc. etc..
Source	The ServiceNow record sys_id on the instance.
Source table	The table the ServiceNow record resides on.
HTTP Status	HTTP return code.
HTTP Response	Entire HTTP response payload.
HTTP Method	ServiceNow REST method label.
Endpoint	The URL endpoint for the REST call.
Active	True or False
Index	Auto-incrementing numeric value.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond REST Payload
    - Name: u_ebond_rest_payload
    - Add module to menu: eBond
4. In the record header, right-click and select **Save**.
5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Reference

    - Column label: Supplier
    - Column name: (this should default to u_supplier)
7. In the **Reference Specification** tab, fill in the following fields:
    - Reference: Company [core_company]
    - Reference qual condition:
        - eBonded is true AND
        - eBond account is not empty
8. Click **Submit**.
9. In the **Columns** table, click **New**.
10. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: Retry count
    - Column name: (this should default to u_retry_count)
11. Under the **Default Value** tab, , fill in the following field:
    - Default value: 0
12. Click **Submit**.
13. In the **Columns** table, click **New**.
14. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: True/False
    - Column label: Retry
    - Column name: (this should default to u_retry)
15. Under the **Default Value** tab, , fill in the following field:
    - Default value: true
16. Click **Submit**.
17. In the **Columns** table, click **New**.
18. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: REST Message
    - Column name: (this should default to u_rest_message)
    - Max length: 40
19. Click **Submit**.
20. In the **Columns** table, click **New**.
21. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Payload
    - Column name: (this should default to u_payload)
    - Max length: 4000
22. Click **Submit**.
23. In the **Columns** table, click **New**.
24. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Format
    - Column name: (this should default to u_format)
    - Max length: 40
25. Click **Submit**.
26. In the **Columns** table, click **New**.

27. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Table Name
    - Column label: Source table
    - Column name: (this should default to u_source_table)
28. Click **Submit**.
29. In the **Columns** table, click **New**.
30. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Document ID
    - Column label: Source
    - Column name: (this should default to u_source)
31. Under **Related Links**, click **Advanced view**.
32. Under the **Dependent Field** tab, click **Use dependent field**.
33. In the **Dependent on field**, select **Source table**.
34. Click **Update**.
35. In the **Columns** table, click **New**.
36. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: HTTP Status
    - Column name: (this should default to u_http_status)
37. Click **Submit**.
38. In the **Columns** table, click **New**.
39. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: HTTP Response
    - Column name: (this should default to u_http_response)
    - Max length: 4000
40. Click **Submit**.
41. In the **Columns** table, click **New**.
42. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: HTTP Method
    - Column name: (this should default to u_http_method)
    - Max length: 40
43. Click **Submit**.
44. In the **Columns** table, click **New**.
45. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Endpoint
    - Column name: (this should default to u_endpoint)
    - Max length: 400
46. Click **Submit**.
47. In the **Columns** table, click **New**.
48. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: True/False
    - Column label: Active
    - Column name: (this should default to u_active)

49. Under **Related Links**, click **Advanced view**.
50. Under the **Calculated Value** tab, fill in the following fields:
    - Calculated: true
    - Calculation:

```
(function calculatedFieldValue(current) {
    var calc = current.u_retry;

    switch (current.u_http_status.toString()) {
        //1×× Informational
        case '100': // Continue
        case '101': // Switching Protocols
        case '102': // Processing
        case '200': // OK
        case '201': // Created
        case '202': // Accepted
        case '203': // Non-authoritative Information
        case '204': // No Content
        case '205': // Reset Content
        case '206': // Partial Content
        case '207': // Multi-Status
        case '208': // Already Reported
        case '226': // IM Used
        case '300': // Multiple Choices
        case '301': // Moved Permanently
        case '302': // Found
        case '303': // See Other
        case '304': // Not Modified
        case '305': // Use Proxy
        case '307': // Temporary Redirect
        case '308': // Permanent Redirect
        case '400': // Bad Request
            calc = false;
            break;
        case '401': // Unauthorized
        case '402': // Payment Required
        case '403': // Forbidden
        case '404': // Not Found
        case '405': // Method Not Allowed
        case '406': // Not Acceptable
        case '407': // Proxy Authentication Required
        case '408': // Request Timeout
        case '409': // Conflict
        case '410': // Gone
        case '411': // Length Required
        case '412': // Precondition Failed
        case '413': // Payload Too Large
        case '414': // Request-URI Too Long
        case '415': // Unsupported Media Type
        case '416': // Requested Range Not Satisfiable
        case '417': // Expectation Failed
        case '418': // I'm a teapot
        case '421': // Misdirected Request
        case '422': // Unprocessable Entity
        case '423': // Locked
```

```
        case '424': // Failed Dependency
        case '426': // Upgrade Required
        case '428': // Precondition Required
        case '429': // Too Many Requests
        case '431': // Request Header Fields Too Large
        case '444': // Connection Closed Without Response
        case '451': // Unavailable For Legal Reasons
        case '499': // Client Closed Request
        case '500': // Internal Server Error
        case '501': // Not Implemented
        case '502': // Bad Gateway
        case '503': // Service Unavailable
        case '504': // Gateway Timeout
            break;
        case '505': // HTTP Version Not Supported
        case '506': // Variant Also Negotiates
        case '507': // Insufficient Storage
        case '508': // Loop Detected
        case '510': // Not Extended
        case '511': // Network Authentication Required
        case '599': // Network Connect Timeout Error
            calc = false;
            break;
        default:
            break;
    }

    // after 48 hours of retrying, turn off active flag
    if (current.u_retry_count >= 48) {
        calc = false;
    }

    return calc; // return the calculated value
})(current);
```

51. Click **Update**.
52. In the **Columns** table, click **New**.
53. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: Index
    - Column name: (this should default to u_index)
    - Read only: True
54. Under **Related Links**, click **Advanced view**.
55. Under the **Default Value** tab, fill in the following field:
    - Use dynamic default: true
    - Dynamic Default value: eBond REST Payload Index Increment
56. Click **Submit**.
57. Click **Update**.

Multi-source Inbound Handling

Inbound incident handling could be setup in an agnostic fashion based on a mutual push-push model setup between the enterprise and suppliers. The mutual push-push model is where the each party sets how they shall receive messages and no party performs a pull/get operation. There are major advantages for this model over one party dictating both how it will send and receive messages.

> **OEM & OSP**
>
> Original equipment manufacturers (OEM) and original service providers (OSP) rarely fit into the mutual push-push model as their clientele is vast and the reliance of the enterprise on the OEMs and OSPs do not warrant a collaborative effort between the enterprise ServiceNow developers and the OEM/OSP. In the case of true partnership between enterprise and suppliers should allow for a mutual push-push model.

The biggest advantage to all parties is long term run and maintain of eBonding operations. If one party changes their fields or data, the other party is not forced into changing their eBonding framework to accommodate; a decision from one party should not impact or cause work for the other. Another advantage is resource utilization. Pull requests are expensive and if an enterprise is being pulled by dozens of suppliers, this places an undue burden on the enterprise. It is far more agreeable that if a change is to occur in one party's record, then that party will pass (push) the delta change to the agreed parties.

Quick Rundown

Suppliers will write to a staging table called *[u_ebond_incident_staging]*. The table will be an extension of the *[sys_import_set_row]* and will have the ability to take advantage of the OOTB ServiceNow transform functionality. The *transform maps* will validate the incoming data and then create or update the appropriate records within the system.

Incident staging table

Normally the REST inbound message handling aspect for eBonding is developed using *Scripted REST API*. For multi-source eBonding a different model will be developed using a staging table. Suppliers will directly write to a staging table instead of a Scripted REST API.

ServiceNow *Transform Maps* provide an elegant and efficient means to do much of what a *Scripted REST API* can do with the use of *Transform Scripts*. The **onStart** is used for preparation of global variables, *onBefore* is used for data validation, and *onAfter* for data operations. All of this efficiently designed by Servicenow in the import set framework to handle inbound REST messages.

> **Staging other tables**
>
> For each type of destination ServiceNow table for multi-source eBonding will require a unique staging table. For example, if the requirement is to have a multi-source eBond with change requests, a similar staging table that that below will need to be created.

The *u_ebond_incident_staging* table holds the field values passed by the supplier for incidents.

The *u_ebond_incident_staging* table will contain the following custom fields:

Field	Description
Work Note	Support work note, customer not visible.
Subcategory	Subcategory for the incident.
State	The state of the incident.
Short Description	The title (or short description) of the incident.
Service Offering	The service offering the incident is impacting.
Service	The service the incident is impacting.
Impact	The impact the incident has on the service.
Urgency	The urgency to resolve the incident.
Sys ID	The sys_id for the local incident ticket.
Number	The number for the local incident ticket.
Hold Reason	The reason the incident ticket was placed on hold.
Description	The description of the incident.
Contact Type	How the incident was reported.
Comment	Customer visible comment.
External Reference	The supplier's incident ticket unique identifier.
External Number	The supplier's incident ticket number.
Error	Returning error information.
CMDB CI	The configuration item the incident is reported against.
Close Notes	Incident closure notes.
Close Code	Incident closure code.
Category	Category for the incident
Caller	The caller who reported the incident.
Assignment Group	The assignment group working the incident.

1. Navigate to **System Definition** > **Tables**.
2. Click **New**.
3. Under the **Table New record** section, fill in the following fields:
    - Label: eBond Incident Staging
    - Name: u_ebond_incident_staging
    - Extends table: Import Set Row [sys_import_set_row]
    - Add module to menu: eBond
4. In the record header, right-click and select **Save**.

5. In the **Columns** table, click **New**.
6. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Work Note
    - Column name: (this should default to u_work_note)
    - Max length: 4000
7. Click **Submit**.
8. In the **Columns** table, click **New**.
9. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Subcategory
    - Column name: (this should default to u_subcategory)
    - Max length: 80
10. Click **Submit**.
11. In the **Columns** table, click **New**.
12. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: State
    - Column name: (this should default to u_state)
    - Max length: 40
13. Click **Submit**.
14. In the **Columns** table, click **New**.
15. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Short Description
    - Column name: (this should default to u_short_description)
    - Max length: 1000
16. Click **Submit**.
17. In the **Columns** table, click **New**.
18. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Service Offering
    - Column name: (this should default to u_service_offering)
    - Max length: 255
19. Click **Submit**.
20. In the **Columns** table, click **New**.
21. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Service
    - Column name: (this should default to u_service)
    - Max length: 255
22. Click **Submit**.
23. In the **Columns** table, click **New**.
24. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Impact

    - Column name: (this should default to u_impact)
    - Max length: 40
25. Click **Submit**.
26. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Urgency
    - Column name: (this should default to u_urgency)
    - Max length: 40
27. Click **Submit**.
28. In the **Columns** table, click **New**.
29. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Sys ID
    - Column name: (this should default to u_sys_id)
    - Max length: 40
30. Click **Submit**.
31. In the **Columns** table, click **New**.
32. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Number
    - Column name: (this should default to u_number)
    - Max length: 40
33. Click **Submit**.
34. In the **Columns** table, click **New**.
35. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Hold Reason
    - Column name: (this should default to u_hold_reason)
    - Max length: 40
36. Click **Submit**.
37. In the **Columns** table, click **New**.
38. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Description
    - Column name: (this should default to u_description)
    - Max length: 4000
39. Click **Submit**.
40. In the **Columns** table, click **New**.
41. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Contact Type
    - Column name: (this should default to u_contact_type)
    - Max length: 40
42. Click **Submit**.
43. In the **Columns** table, click **New**.
44. In the **Dictionary Entry New record** section, fill in the following fields:

    - Type: String
    - Column label: Comment
    - Column name: (this should default to u_comment)
    - Max length: 4000
45. Click **Submit**.
46. In the **Columns** table, click **New**.
47. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: External Reference
    - Column name: (this should default to u_external_reference)
    - Max length: 40
48. Click **Submit**.
49. In the **Columns** table, click **New**.
50. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: External Number
    - Column name: (this should default to u_external_number)
    - Max length: 40
51. Click **Submit**.
52. In the **Columns** table, click **New**.
53. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Error
    - Column name: (this should default to u_error)
    - Max length: 4000
54. Click **Submit**.
55. In the **Columns** table, click **New**.
56. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: CMDB CI
    - Column name: (this should default to u_cmdb_ci)
    - Max length: 40
57. Click **Submit**.
58. In the **Columns** table, click **New**.
59. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Close Notes
    - Column name: (this should default to u_close_notes)
    - Max length: 1000
60. Click **Submit**.
61. In the **Columns** table, click **New**.
62. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Close Code
    - Column name: (this should default to u_close_code)
    - Max length: 40

63. Click **Submit**.
64. In the **Columns** table, click **New**.
65. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Category
    - Column name: (this should default to u_category)
    - Max length: 80
66. Click **Submit**.
67. In the **Columns** table, click **New**.
68. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Caller
    - Column name: (this should default to u_caller)
    - Max length: 40
69. Click **Submit**.
70. In the **Columns** table, click **New**.
71. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: Assignment Group
    - Column name: (this should default to u_assignment_group)
    - Max length: 40
72. Click **Submit**.
73. Click **Update**.

Inbound Transform

The inbound *transform map* uses *transform scripts* to perform various operations on the data directly passed by the supplier. The *transform scripts* can be set to *when* they are executed against the data in the transform.

- **onStart:** Sets up the global variables used by the other transform scripts. Global variables will be capitalized for ease of development.
- **onBefore:** Validates security and the data passed by the supplier. The order of scripts are important as pre-checks must complete before continuing the transform.
- **on:After** Creates or updates the needed fields within the records.
- **on:Complete** Verifies all operations and updates return information to the supplier.

1. Navigate to **System Import Sets** > **Transform Maps**.
2. Click **New**.
3. In the **Table Transform Map New record** section, fill in the following fields:
    - Name: eBond Incident Transform
    - Source table: eBond Incident Staging [u_ebond_incident_staging]
    - Target table: eBond Relationship [u_ebond_relationship]
4. In the record header, right-click and select **Save**.
5. In the **Field Maps** tab, click **New**.

6. In the **Field Map New record** section, fill in the following fields:
    - Source field: External Number [u_external_number]
    - Target field: Correlation Number [u_correlation_number]
    - Coalesce: True
7. Click **Submit**.

> **Coalesce field not indexed**
>
> If a pop-up box appears titled *Coalesce field not indexed*, then click **OK**, fill in the followup contact information and click **OK**.

8. In the **Field Maps** tab, click **New**.
9. In the **Field Map New record** section, fill in the following fields:
    - Source field: External reference [u_external_reference]
    - Target field: Correlation ID [u_correlation_id]
    - Coalesce: false
10. Click **Submit**.
11. In the **Field Maps** tab, click **New**.
12. In the **Field Map New record** section, click **Use source script** and fill in the following fields:
    - Choice action: ignore
    - Target field: Company [u_company]
    - Coalesce field: true
    - Script:

```
answer = (function transformEntry(source) {

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Field Map] Company';
    if (SUPPLIER == undefined) {
        eLog.u_supplier = 'Unknown';
    } else {
        eLog.u_supplier = SUPPLIER;
    }
    eLog.write('Info', 'Entering.');

    // coalesce script is called at various times in the transform
    // ignore the time(s) when the row is not even being evaluated
    if (source.sys_created_by == "") {
        eLog.write('Info', 'Exiting.');
        return;
    }
```

```
    // coalesce script is called before ServiceNow global transform map
    // variables are defined
    if (error == undefined) {
        var error = "";
    }

    // once the source row is read in during the transform, the
    // global error variable is defined by ServiceNow, and to
    // prevent the coalesce script to be called twice on error,
    // this check is required
    if (error == true) {
        eLog.write('Info', 'Exiting.');
        return;
    }

    // find the company the account is associated with
    var user = new GlideRecord('sys_user');
    user.addQuery('user_name', source.sys_created_by);
    user.query();
    if (!user.next()) {
        // supplier is not eBond enabled
        eLog.write('High', 'Security Violation: An account (' + source.sys_

        error = true;
        error_message = "Security violation.";
        source.u_error = error_message;

        eLog.write('Info', 'Exiting.');
        return "";
    }

    // find the company the account is associated with
    var company = new GlideRecord('core_company');
    company.addQuery('u_ebond_account', user.sys_id);
    company.query();
    if (company.next()) {
        // make sure the supplier is still eBond approved
        if (company.u_ebonded) {
            eLog.write('Info', 'Exiting.');
            return company.sys_id;
        }

        // supplier is not eBond enabled
        eLog.write('Low', 'Security Violation: The supplier ' + company.nam

        error = true;
        error_message = "Security violation.";
        source.u_error = error_message;

        eLog.write('Info', 'Exiting.');
        return "";
    }

    // creator is not associated with a supplier
    eLog.write('High', 'Security Violation: An eBond account (' + source.sy

    error = true;
```

```
    error_message = "Security violation.";
    source.u_error = error_message;

    eLog.write('Info', 'Exiting.');
    return ""; // return the value to be put into the target field

})(source);
```

13. Click **Submit**.

> **Coalesce fields**
>
> Only the *target field* fields *u_correlation_number* and *u_company* fields are coalesce fields. The *u_company* is derived internally from the *sys_created_by* field that holds the account that wrote the record. This removes the means to supplant company information. The *u_correlation_number* by itself is not sufficient to make it unique enough with multiple eBonded suppliers. The combination of the *u_correlation_number* and *u_company* is required; as the assumption is the supplier will not duplicate their ticket numbers. However, two or more suppliers could have the same ticket numbers unrelated to each other. The reason that the *u_external_reference* is not used is it's typically not a required field. Some ticket solutions do not use reference qualifiers and the ticket number is sufficient when passing information back to the supplier.

14. In the **Transform Scripts** tab, click **New**.
15. In the **Transform Script New record** section, fill in the following fields:
    - When: onStart
    - Order: 1
    - Script:

```
// Initialize global variables
var eLog = new eBondLog();
eLog.u_direction = 'inbound';
eLog.u_location = 'Transform Maps';
eLog.u_name = 'eBond Incident Transform';
eLog.u_source = '[Transform Script] Initialize global variables';
eLog.u_supplier = 'Unknown';
eLog.write('Info', 'Entering.');


var SUPPLIER = ""; // supplier's name
// SUPPLIER: the string value that represents the supplier
//   used primarily for logging and registry operations

var COMPANY = ""; // supplier's company record
// COMPANY: the sys_id of the local [core_company] table record dor the sup

var INCIDENT = ""; // incident reference
var INCIDENT_NUMBER = ""; // incident number
```

```
// INCIDENT: the sys_id of the local [incident] table record
// INCIDENT_NUMBER: the number of the local [incident] table record

var URL = ""; // URL reference back to supplier ticket
// URL: the reference back to the supplier's ticket

// action: is a OOTSB ServiceNow variable to represent the creation of a ne

var OPERATION = "invalid";
// OPERATION: what action to take on the [incident] table record
//   create - create a new incident record
//   update - update an existing incident record

var EXECUTION = "invalid";
// EXECUTION: what the operation will do
//   reflect - changes from the supplier are mapped to the fields in the in
//   inform - changes from the supplier are mapped to the work notes in the
//   debond - deBonds the supplier and enterprise ticket.

var RELATIONSHIP = "";

var SUBCATEGORY = "Support";
var STATE = "1";
var STATE_STR = "New";
var SERVICE_OFFERING = "";
var SERVICE = "";
var IMPACT = "3";
var IMPACT_STR = "3 - Low";
var URGENCY = "3";
var URGENCY_STR = "3 - Low";
var HOLD_REASON = "";
var HOLD_REASON_STR = "";
var CONTACT_TYPE = "ebond";
var CLOSE_CODE = "";
var CATEGORY = "Managed Service";
var CALLER = "";
var ASSIGNMENT_GROUP = "";

eLog.write('Info', 'Exiting.');
```

16. Click **Submit**.
17. In the **Transform Scripts** tab, click **New**.
18. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 5
    - Script:

```
// SUPPLIER
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
```

```
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] SUPPLIER';
    eLog.u_supplier = 'Unknown';
    eLog.write('Info', 'Entering.');

    var registry = new GlideRecord('u_ebond_registry');
    registry.addQuery('u_key', 'inbound.account');
    registry.addQuery('u_value', source.sys_created_by);
    registry.query();
    if (registry.next()) {
        SUPPLIER = registry.u_supplier;

        eLog.u_supplier = SUPPLIER;
        eLog.write('Info', 'SUPPLIER = ' + SUPPLIER);
    } else {
        eLog.u_supplier = 'Unknown';
        eLog.write('Medium', 'Registry Error: An eBond account (' + source

        error = true;
        error_message = "Unregistered supplier account.";
        source.u_error = error_message;
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

19. Click **Submit**.
20. In the **Transform Scripts** tab, click **New**.
21. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 10
    - Script:

```
// COMPANY
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] COMPANY';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    // find the user account that created the record
    var user = new GlideRecord('sys_user');
    user.addQuery('user_name', source.sys_created_by);
    user.query();
    if( user.next() ) {
        CALLER = user.sys_id;
```

```
        eLog.write('Info', 'CALLER = ' + CALLER);
    }

    // find the company the account is associated with
    var company = new GlideRecord('core_company');
    company.addQuery('u_ebond_account', user.sys_id);
    company.query();
    if (company.next()) {
        COMPANY = company.sys_id;
        eLog.write('Info', 'COMPANY = ' + COMPANY);
    }

    eLog.write('Info', 'Exiting.');
})(source);
```

22. Click **Submit**.
23. In the **Transform Scripts** tab, click **New**.
24. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 20
    - Script:

```
// RELATIONSHIP
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] RELATIONSHIP';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    if (action == 'insert') {
        eLog.write('Info', 'Exiting.');
        return;
    }

    var relationship = new GlideRecord('u_ebond_relationship');
    relationship.addQuery('u_correlation_number',source.u_external_number);
    relationship.addQuery('u_company', COMPANY);
    relationship.query();
    if (relationship.next()) {
        RELATIONSHIP = relationship.sys_id;
        eLog.write('Info', 'RELATIONSHIP = ' + RELATIONSHIP);
    } else {
        eLog.write('High', 'Transform action is update, but cannot find eBo
        error = true;
        error_message = "Cannot find eBond relationship.";
        source.u_error = error_message;
    }
```

```
    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

25. Click **Submit**.
26. In the **Transform Scripts** tab, click **New**.
27. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 30
    - Script:

```
// OPERATION & EXECUTION
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] OPERATION & EXECUTION';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    /*
              scenario matrix
              invalid is where logically the transform cannot handle the sc
              if OPERATION or EXECUTION found to default to invalid will lo
              action is the operation the transform will have on the u_ebon

              action  |  OPERATION  |  EXECUTION  |  SCENARIO
              ------------------------------------------------------------
              insert  |  create     |  reflect    | new eBond from supplier
              insert  |  create     |  inform     |   invalid - cannot crea
              insert  |  create     |  debond     |   invalid - cannot crea
              insert  |  update     |  reflect    |   invalid - ticket woul
              insert  |  update     |  inform     | new eBond from supplier
              insert  |  update     |  debond     |   invalid - cannot crea
              update  |  create     |  reflect    |   invalid - supplier ca
              update  |  create     |  inform     |   invalid - cannot crea
              update  |  create     |  debond     |   invalid - cannot crea
              update  |  update     |  reflect    | supplier is updating a
              update  |  update     |  inform     | supplier is updating a
                                                  | supplier is re-eBonding
              update  |  update     |  debond     | supplier is debonding t
            */

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ------------------------------------------------------------
    //  insert  |  create     |  reflect    | new eBond from supplier
    if (action == 'insert' && source.u_number == '' && source.u_sys_id ==
        OPERATION = 'create';
        EXECUTION = 'reflect';
```

```
        eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
        eLog.write('Info', 'Exiting.');
        return;
    }

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ---------------------------------------------------------------------
    //  insert  |  update     |  inform     | new eBond from supplier to ex
    if (action == 'insert' && (source.u_number != '' || source.u_sys_id !=
        var incident = new GlideRecord('incident');
        if (source.u_number != '') {
            incident.addQuery('number', source.u_number);
        } else {
            incident.addQuery('sys_id', source.u_sys_id);
        }
        incident.query();
        if (incident.next()) {
            OPERATION = 'update';
            EXECUTION = 'inform';
            eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTION =
            eLog.write('Info', 'Exiting.');
            return;
        } else {
            if (source.u_number != '') {
                eLog.write('Medium', 'Could not find incident ' + source.u_
            } else {
                eLog.write('Medium', 'Could not find the incident reference
            }

            var relationship = new eBondRelationship();
            relationship.updateIn(RELATIONSHIP,'down');

            error = true;
            error_message = "Incident not found.";
            source.u_error = error_message;

            eLog.write('Info', 'Exiting.');
            return;
        }
    }

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ---------------------------------------------------------------------
    //  update  |  *          |  *          |  if the eBond relationship ex
    //                                      |  if the incident is closed th
    if (action == 'update') {
        var incident = new GlideRecord('incident');
        incident.addQuery('number', relationship.u_source);
        incident.query();
        if (incident.next()) {
            if (incident.getValue('state') == 7) { // closed
                error = true;
                error_message = "Incident " + incident.number + "cannot be
                source.u_error = error_message;

                eLog.write('Info', error_mnessage);
                eLog.write('Info', 'Exiting');
```

```
                return;
            }
        }
    }

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ---------------------------------------------------------------------
    //  update  |  update     |  debond     | supplier is debonding the tic
    if (action == 'update' && (source.u_number == '-1' || source.u_sys_id =
        OPERATION = 'update';
        EXECUTION = 'debond';

        eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
        eLog.write('Info', 'Exiting.');
        return;
    }

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ---------------------------------------------------------------------
    //  update  |  update     |  inform     | supplier is re-eBonding to a
    //
    if (action == 'update' && target.u_status.indexOf('debonded') != -1) {
        OPERATION = 'update';
        EXECUTION = 'inform';

        eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
        eLog.write('Info', 'Exiting.');
        return;
    }

    //  action  |  OPERATION  |  EXECUTION  |  SCENARIO
    //  ---------------------------------------------------------------------
    //  update  |  update     |  reflect    | supplier is updating a ticket
    //  update  |  update     |  inform     | supplier is updating a ticket
    if (action == 'update' && target.u_status.indexOf('debonded') == -1) {
        var relationship = new GlideRecord('u_ebond_relationship');
        relationship.addQuery('u_company', COMPANY);
        relationship.addQuery('u_correlation_number', source.u_external_nur
        relationship.query();
        if (relationship.next()) {
            if (relationship.u_reflect == true) {
                OPERATION = 'update';
                EXECUTION = 'reflect';

                eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTIC
                eLog.write('Info', 'Exiting.');
                return;
            } else {
                OPERATION = 'update';
                EXECUTION = 'inform';

                eLog.write('Info', 'OPERATION = ' + OPERATION + '\nEXECUTIC
                eLog.write('Info', 'Exiting.');
                return;
            }
        } else {
            error = true;
```

```
            error_message = "Internal error, eBond reference not found with
            source.u_error = error_message;

            eLog.write('High', 'Internal Error: eBond reference not found w
            eLog.write('info', 'Exiting.');
            return;
        }
    }

    // at this point all valid logic paths have been tested.
    var relationship = new eBondRelationship();
    relationship.updateIn(RELATIONSHIP,'down');

    error = true;
    error_message = "Internal error, logic path exhaustion.";
    source.u_error = error_message;

    eLog.write('High', 'Internal Error: Logic path exhaustion; OPERATION [
    eLog.write('Info', 'Exiting.');
    return;

})(source, map, log, target);
```

28. Click **Submit**.
29. In the **Transform Scripts** tab, click **New**.
30. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 40
    - Script:

```
// check required fields
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] check required fields';
    eLog.u_supplier = 'Unknown';
    eLog.write('Info', 'Entering.');

    var missingField = false;
    var missingFields = 'Missing the folowing required field(s): ';
    var separator = '';

    // supplier ticket number is mandatory in all cases
    if (source.u_external_number == '') {
        missingField = true;
        missingFields = missingFields + separator + 'u_external_number';
        separator = ', ';
    }
```

```
    // if new relationship, provide a short description
    if (action == 'insert' && OPERATION == 'create' && source.u_short_descr
        missingField = true;
        missingFields = missingFields + separator + 'u_short_description';
        separator = ', ';
    }

    // if new relationship, provide a description
    if (action == 'insert' && OPERATION == 'create' && source.u_description
        missingField = true;
        missingFields = missingFields + separator + 'u_description';
        separator = ', ';
    }

    // if state is on hold, provide a hold reason
    if (source.u_state == '3' && source.u_hold_reason == '') {
        missingField = true;
        missingFields = missingFields + separator + 'u_hold_reason';
        separator = ', ';
    }

    // if state is resolved or closed, provide a close code
    if ((source.u_state == '6' || source.u_state == '7') && source.u_close_
        missingField = true;
        missingFields = missingFields + separator + 'u_close_code';
        separator = ', ';
    }

    // if state is resolved or closed, provide a close notes
    if ((source.u_state == '6' || source.u_state == '7') && source.u_close_
        missingField = true;
        missingFields = missingFields + separator + 'u_close_notes';
        separator = ', ';
    }

    if (missingField) {
        missingFields = missingFields + '.';
        eLog.write('Low',missingFields + ' Reference: ' + source.sys_id);

        var relationship = new eBondRelationship();
        relationship.updateIn(RELATIONSHIP,'down');
        error = true;
        error_message = missingFields;
        source.u_error = error_message;
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

31. Click **Submit**.

32. In the **Transform Scripts** tab, click **New**.

33. In the **Transform Script New record** section, fill in the following fields:

- When: onBefore
- Order: 50
- Script:

```
// INCIDENT
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] INCIDENT';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    if ((source.u_number != '' || source.u_sys_id != '') && (source.u_numbe
        var incident = new GlideRecord('incident');
        if (source.u_number != '') {
            incident.addQuery('number', source.u_number);
        } else {
            incident.addQuery('sys_id', source.u_sys_id);
        }
        incident.query();
        if (incident.next()) {
            INCIDENT = incident.sys_id;
            INCIDENT_NUMBER = incident.number;
            eLog.write('Info', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + '\r
            eLog.write('Info', 'Exiting.');
            return;
        } else {
            if (source.u_number != '') {
                eLog.write('Medium', 'Could not find incident ' + source.u_
            } else {
                eLog.write('Medium', 'Could not find the incident reference
            }

            var relationship = new eBondRelationship();
            relationship.updateIn(RELATIONSHIP,'down');

            error = true;
            error_message = "Incident not found.";
            source.u_error = error_message;

            eLog.write('Info', 'Exiting.');
            return;
        }
        // the supplier did not pass local ticket information
    } else if (action == 'update') {
        var relationship = new GlideRecord('u_ebond_relationship');
        if (source.u_external_number != '') {
            relationship.addQuery('u_correlation_number', source.u_external
        } else {
            relationship.addQuery('u_correlation_id', source.u_external_ref
        }
```

```
        relationship.query();
        if (relationship.next()) {
            INCIDENT = relationship.u_source.sys_id;
            INCIDENT_NUMBER = relationship.u_source.number;
            eLog.write('Info', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + '\r
            eLog.write('Info', 'Exiting');
            return;
        } else {
            if (source.u_external_number != '') {
                eLog.write('High', 'Could not find external correlation nur
            } else {
                eLog.write('High', 'Could not find the external correlatior
            }

            error = true;
            error_message = "eBond relationship not found.";
            source.u_error = error_message;

            eLog.write('Info', 'Exiting');
            return;
        }
    }

    eLog.write('Info', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + '\nINCIDENT
    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

34. Click **Submit**.

35. In the **Transform Scripts** tab, click **New**.

36. In the **Transform Script New record** section, fill in the following fields:

    - When: onBefore
    - Order: 60
    - Script:

```
// URL
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] URL';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    // deBond check
    if (EXECUTION == 'debond') {
        eLog.write('Info', 'Exiting.');
        return;
    }
```

```
    var registry = new GlideRecord('u_ebond_registry');
    registry.addQuery('u_supplier', SUPPLIER);
    registry.addQuery('u_key', 'incident.external.url');
    registry.query();
    if (registry.next()) {
        var tRef = registry.u_value;

        var tRef1 = tRef.replace('CORRELATION_ID', source.u_external_refere
        URL = tRef1.replace('CORRELATION_NUMBER', source.u_external_number)

        eLog.write('Info', 'URL = ' + URL);
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

37. Click **Submit**.

38. In the **Transform Scripts** tab, click **New**.

39. In the **Transform Script New record** section, fill in the following fields:

    - When: onBefore
    - Order: 200
    - Script:

```
// validate data
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] validate data';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    var invalidData = false;
    var invalidInfo = 'Invalid values for the field(s): ';
    var separator = '';

    if (source.u_sys_id != '' && source.u_sys_id != '-1') {
        var incident = new GlideRecord("incident");
        incident.addQuery("sys_id", source.u_sys_id);
        incident.query();
        if (incident.next()) {
            INCIDENT = incident.sys_id;
            eLog.write('Info', 'INCIDENT = ' + INCIDENT);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_sys_id';
            separator = ', ';
        }
    }
```

```
    if (source.u_subcategory != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "subcategory");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_subcategory);
        data_map.query();
        if (data_map.next()) {
            SUBCATEGORY = data_map.u_source_value;
            eLog.write('Info', 'SUBCATEGORY = ' + SUBCATEGORY);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_subcategory';
            separator = ', ';
        }
    }

    if (source.u_state != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "state");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_state);
        data_map.query();
        if (data_map.next()) {
            STATE = data_map.u_source_value;
            STATE_STR = data_map.u_note;
            eLog.write('Info', 'STATE = ' + STATE + ' (' + STATE_STR + ')')
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_state';
            separator = ', ';
        }
    }

    if (source.u_service_offering != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "service_offering");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_service_offering);
        data_map.query();
        if (data_map.next()) {
            SERVICE_OFFERING = data_map.u_source_value;
            eLog.write('Info', 'SERVICE_OFFERING = ' + SERVICE_OFFERING);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_service_offering';
            separator = ', ';
        }
    } else if (OPERATION == 'create') {
        var eRegistry = new GlideRecord("u_ebond_registry");
        eRegistry.addQuery("u_supplier", SUPPLIER);
```

```
        eRegistry.addQuery("u_key", "inbound.service.offering");
        eRegistry.query();
        if (eRegistry.next()) {
            SERVICE_OFFERING = eRegistry.u_value;
            eLog.write('Info', 'SERVICE_OFFERING = ' + SERVICE_OFFERING);
        } else {
            var relationship = new eBondRelationship();
            relationship.updateIn(RELATIONSHIP,'down');

            error = true;
            error_message = "Internal error, missing registry key inbound.s
            source.u_error = error_message;

            eLog.write('High', 'Internal Error: Missing registry key inbour
            eLog.write('Info', 'Exiting.');
            return;
        }
    }

    if (source.u_service != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "service");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_service);
        data_map.query();
        if (data_map.next()) {
            SERVICE = data_map.u_source_value;
            eLog.write('Info', 'SERVICE = ' + SERVICE);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_service';
            separator = ', ';
        }
    } else if (OPERATION == 'create') {
        var eRegistry = new GlideRecord("u_ebond_registry");
        eRegistry.addQuery("u_supplier", SUPPLIER);
        eRegistry.addQuery("u_key", "inbound.service");
        eRegistry.query();
        if (eRegistry.next()) {
            SERVICE = eRegistry.u_value;
            eLog.write('Info', 'SERVICE = ' + SERVICE);
        } else {
            eLog.write('High', 'Exiting. Missing inbound.service registry k

            var relationship = new eBondRelationship();
            relationship.updateIn(RELATIONSHIP,'down');

            error = true;
            error_message = "Internal error, missing registry key inbound.s
            source.u_error = error_message;

            wLog.write('Info', 'Exiting.');
            return;
        }
    }
```

```
    if (source.u_impact != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "impact");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_impact);
        data_map.query();
        if (data_map.next()) {
            IMPACT = data_map.u_source_value;
            IMPACT_STR = data_map.u_note;
            eLog.write('Info', 'IMPACT = ' + IMPACT + ' (' + IMPACT_STR +
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_impact';
            separator = ', ';
        }
    }

    if (source.u_urgency != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "urgency");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_urgency);
        data_map.query();
        if (data_map.next()) {
            URGENCY = data_map.u_source_value;
            URGENCY_STR = data_map.u_note;
            eLog.write('Info', 'URGENCY = ' + URGENCY + ' (' + URGENCY_STR
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_urgency';
            separator = ', ';
        }
    }

    if (source.u_number != '' && source.u_number != '-1' ) {
        var incident = new GlideRecord("incident");
        incident.addQuery("number", source.u_number);
        incident.query();
        if (incident.next()) {
            INCIDENT = incident.sys_id;
            eLog.write('Info', 'INCIDENT = ' + INCIDENT);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_number';
            separator = ', ';
        }
    }

    if (source.u_hold_reason != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
```

```
        data_map.addQuery("u_classification", "hold_reason");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_hold_reason);
        data_map.query();
        if (data_map.next()) {
            HOLD_REASON = data_map.u_source_value;
            HOLD_REASON_STR = data_map.u_note;
            eLog.write('Info', 'HOLD_REASON = ' + HOLD_REASON + ' (' + HOLD
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_hold_reason';
            separator = ', ';
        }
    }

    if (source.u_contact_type != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "contact_type");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_contact_type);
        data_map.query();
        if (data_map.next()) {
            CONTACT_TYPE = data_map.u_source_value;
            eLog.write('Info', 'CONTACT_TYPE = ' + CONTACT_TYPE);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_contact_type';
            separator = ', ';
        }
    }

    if (source.u_close_code != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "close_code");
        data_map.addQuery("u_direction", "inbound");
        data_map.addQuery("u_supplier_value", source.u_close_code);
        data_map.query();
        if (data_map.next()) {
            CLOSE_CODE = data_map.u_source_value;
            eLog.write('Info', 'CLOSE_CODE = ' + CLOSE_CODE);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_close_code';
            separator = ', ';
        }
    }

    if (source.u_category != '') {
        var data_map = new GlideRecord("u_ebond_data_map");
        data_map.addQuery("u_supplier", "All");
        data_map.addQuery("u_module", "incident");
        data_map.addQuery("u_classification", "category");
        data_map.addQuery("u_direction", "inbound");
```

```
        data_map.addQuery("u_supplier_value", source.u_category);
        data_map.query();
        if (data_map.next()) {
            CATEGORY = data_map.u_source_value;
            eLog.write('Info', 'CATEGORY = ' + CATEGORY);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_category';
            separator = ', ';
        }
    }

    if (source.u_caller != '') {
        var caller = new GlideRecord("sys_user");
        caller.addQuery("email", source.u_caller);
        caller.query();
        if (caller.next()) {
            CALLER = caller.sys_id;
            eLog.write('Info', 'CALLER = ' + CALLER);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_caller';
            separator = ', ';
        }
    }

    if (source.u_assignment_group != '') {
        var assignmentGroup = new GlideRecord("sys_user_group");
        assignmentGroup.addQuery("name", source.u_assignment_group);
        assignmentGroup.query();
        if (assignmentGroup.next()) {
            ASSIGNMENT_GROUP = assignmentGroup.sys_id;
            eLog.write('Info', 'ASSIGNMENT_GROUP = ' + ASSIGNMENT_GROUP);
        } else {
            invalidData = true;
            invalidInfo = invalidInfo + separator + 'u_assignment_group';
            separator = ', ';
        }
    } else if (OPERATION == 'create') {
        var eRegistry = new GlideRecord("u_ebond_registry");
        eRegistry.addQuery("u_supplier", SUPPLIER);
        eRegistry.addQuery("u_key", "inbound.assignment.group");
        eRegistry.query();
        if (eRegistry.next()) {
            ASSIGNMENT_GROUP = eRegistry.u_value;
            eLog.write('Info', 'ASSIGNMENT_GROUP = ' + ASSIGNMENT_GROUP);
        } else {
            eLog.write('High', 'Missing inbound.assignment.group default fo

            var relationship = new eBondRelationship();
            relationship.updateIn(RELATIONSHIP,'down');

            error = true;
            error_message = "Internal error, missing registry value inbound
            source.u_error = error_message;

            eLog.write('Info', 'Exiting');
```

```
            return;
        }
    }

    if (invalidData) {
        invalidInfo = invalidInfo + '.';
        eLog.write('Low', invalidInfo + ' Reference: ' + source.sys_id);

        var relationship = new eBondRelationship();
        relationship.updateIn(RELATIONSHIP,'down');

        error = true;
        error_message = invalidInfo;
        source.u_error = error_message;
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

40. Click **Submit**.
41. In the **Transform Scripts** tab, click **New**.
42. In the **Transform Script New record** section, fill in the following fields:
    - When: onBefore
    - Order: 210
    - Script:

```
// deBond Relationship
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] deBond Relationship';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    // eBond check
    if (EXECUTION != 'debond') {
        eLog.write('Info', 'Exiting.');
        return;
    }

    // deBond the relationship
    target.u_status = 'debonded';
    target.u_in = 'up';

    // retrieve the incident record to update
    var incident = new GlideRecord('incident');
    incident.addQuery('sys_id', INCIDENT);
    incident.query();
```

```
    incident.next();

    // work notes in the incident vary depending on existing relationships
    var workNotes = '';

    // check to see if there are any other suppliers tickets ebonded
    var relationship = new GlideRecord('u_ebond_relationship');
    var advQuery = 'u_company=' + COMPANY + '^u_source=' + INCIDENT + '^u_s
    relationship.addEncodedQuery(advQuery);
    relationship.query();
    // there are other supplier tickets eBonded to this
    if (relationship.next()) {
        workNotes = SUPPLIER + ' has requested to deBond their ticket ' + s

        eLog.write('Info', 'deBond TBD - NOT COMPLETED.');
    } else {
        // there are no other tickets eBonded from the supplier to this tic
        // remove the reference from the incident 'eBonded with' field
        workNotes = SUPPLIER + ' has requested to deBond their ticket ' + s

        var arrUtil = new ArrayUtil();
        var arr = incident.u_ebonded_with.toString().split(',');
        var pos = arrUtil.indexOf(arr, COMPANY);
        if (pos >= 0) {
            arr.splice(pos, 1);
            incident.u_ebonded_with = arr.toString();
            incident.update();
            eLog.write('Info', 'Removed company record ' + COMPANY + ' from
        }

        eLog.write('Info', 'deBond TBD - NOT COMPLETED.');
    }

    // set the proper work notes for the incident and update the incident r
    incident.work_notes = workNotes;
    incident.update();

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

43. Click **Submit**.

44. In the **Transform Scripts** tab, click **New**.

45. In the **Transform Script New record** section, fill in the following fields:

    - When: onBefore
    - Order: 220
    - Script:

```
// eBond Relationship
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
```

```
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] eBond Relationship';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    // deBond check
    if (EXECUTION == 'debond') {
        eLog.write('Info', 'Exiting.');
        return;
    }

    // create a new incident record
    if (OPERATION == 'create' && INCIDENT == '') {
        var grInc = new GlideRecord("incident");
        grInc.initialize();
        grInc.insert();
        grInc.subcategory = SUBCATEGORY;
        grInc.state = STATE;
        grInc.service_offering = SERVICE_OFFERING;
        grInc.business_service = SERVICE;
        grInc.impact = IMPACT;
        grInc.urgency = URGENCY;
        grInc.contact_type = CONTACT_TYPE;
        grInc.category = CATEGORY;
        grInc.caller = CALLER;
        grInc.assignment_group = ASSIGNMENT_GROUP;
        grInc.update();
        INCIDENT_NUMBER = grInc.number;
        INCIDENT = grInc.sys_id.toString();
        eLog.write('Info', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + ' INCID
    }

    // regardless update the relationship
    // this will capture re-bond'ing incidents too
    target.u_url = URL;
    target.u_status = 'ebonded';
    target.u_state = STATE;
    target.u_in = 'up';
    target.u_source_table = 'incident';
    target.u_source = INCIDENT;
    if (EXECUTION == 'reflect') {
        target.u_reflect = true;
    } else {
        target.u_reflect = false;
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

46. Click **Submit**.

47. In the **Transform Scripts** tab, click **New**.

48. In the **Transform Script New record** section, fill in the following fields:

    - When: onBefore
    - Order: 230
    - Script:

```
// update incident
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] update incident';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering');

    if (EXECUTION == 'debond') {
        eLog.write('Info', 'Exiting.');
        return;
    }

    // see transform script 'eBond Relationship' when OPERATION is create
    // the incident is created there so that the relationship record is pop
    // INCIDENT global variable shall be defined at this point in the trans

    // if EXECUTION is inform, then field changes are stored as work notes
    var notes = SUPPLIER + ' has updated the following fields on their tick
    var noteFlag = false;
    var incidentFlag = false;

    var incident = new GlideRecord('incident');
    //incident.addQuery('sys_id', INCIDENT);
    incident.addQuery('number', INCIDENT_NUMBER);
    incident.query();
    if (incident.next()) {

        if (source.u_work_note != '') {
            incident['work_notes'].setJournalEntry(source.u_work_note);
            incidentFlag = true;
            eLog.write('Info', 'Added work notes. ' + source.u_work_note);
        }

        if (source.u_subcategory != '') {
            if (EXECUTION == 'reflect') {
                incident.subcategory = SUBCATEGORY;
                incidentFlag = true;
                eLog.write('Info', 'Set subcategory.');
            } else {
                notes = notes + '\nSubcategory: ' + SUBCATEGORY;
                noteFlag = true;
                eLog.write('Info', 'Added subcategory to work notes.');
            }
        }
```

```
        if (source.u_state != '') {
            if (EXECUTION == 'reflect') {
                incident.state = STATE;
                incidentFlag = true;
                eLog.write('Info', 'Set state.');
            } else {
                notes = notes + '\nState: ' + STATE_STR;
                noteFlag = true;
                eLog.write('Info', 'Added state to work notes.');
            }
        }

        if (source.u_short_description != '') {
            if (EXECUTION == 'reflect') {
                incident.short_description = source.u_short_description;
                incidentFlag = true;
                eLog.write('Info', 'Set short description.');
            } else {
                notes = notes + '\nShort description: ' + source.u_short_de
                noteFlag = true;
                eLog.write('Info', 'Added short description to work notes.
            }
        }

        if (source.u_service_offering != '') {
            if (EXECUTION == 'reflect') {
                incident.service_offering = SERVICE_OFFERING;
                incidentFlag = true;
                eLog.write('Info', 'Set service offering.');
            } else {
                notes = notes + '\nService offering: ' + SERVICE_OFFERING;
                noteFlag = true;
                eLog.write('Info', 'Added service offering to work notes.')
            }
        }

        if (source.u_service != '') {
            if (EXECUTION == 'reflect') {
                incident.service = SERVICE;
                incidentFlag = true;
                eLog.write('Info', 'Set service.');
            } else {
                notes = notes + '\nService: ' + SERVICE;
                noteFlag = true;
                eLog.write('Info', 'Added service to work notes.');
            }
        }

        if (source.u_impact != '') {
            if (EXECUTION == 'reflect') {
                incident.impact = IMPACT;
                incidentFlag = true;
                eLog.write('Info', 'Set impact.');
            } else {
                notes = notes + '\nImpact: ' + IMPACT_STR;
                noteFlag = true;
                eLog.write('Info', 'Added impact to work notes.');
```

```
            }
        }

        if (source.u_urgency != '') {
            if (EXECUTION == 'reflect') {
                incident.urgency = URGENCY;
                incidentFlag = true;
                eLog.write('Info', 'Set urgency.');
            } else {
                notes = notes + '\nUrgency: ' + URGENCY_STR;
                noteFlag = true;
                eLog.write('Info', 'Added urgency to work notes.');
            }
        }

        if (source.u_hold_reason != '') {
            if (EXECUTION == 'reflect') {
                incident.hold_reason = HOLD_REASON;
                incidentFlag = true;
                eLog.write('Info', 'Set hold reason.');
            } else {
                notes = notes + '\nHold reason: ' + HOLD_REASON_STR;
                noteFlag = true;
                eLog.write('Info', 'Added hold reason to work notes.');
            }
        }

        if (source.u_description != '') {
            if (EXECUTION == 'reflect') {
                incident.description = source.u_description;
                incidentFlag = true;
                eLog.write('Info', 'Set description.');
            } else {
                notes = notes + '\nDescription: ' + source.u_description;
                noteFlag = true;
                eLog.write('Info', 'Added description to work notes.');
            }
        }

        if (source.u_contact_type != '') {
            if (EXECUTION == 'reflect') {
                incident.contact_type = CONTACT_TYPE;
                incidentFlag = true;
                eLog.write('Info', 'Set contact type.');
            } else {
                notes = notes + '\nContact type: ' + CONTACT_TYPE;
                noteFlag = true;
                eLog.write('Info', 'Added contact type to work notes.');
            }
        }

        if (source.u_comment != '') {
            incident['comments'].setJournalEntry(source.u_comment);
            incidentFlag = true;
            eLog.write('Info', 'Added comment.');
        }
```

```
        if (source.u_cmdb_ci != '') {
            notes = notes + '\n\tCMDB CI: ' + source.u_cmdb_ci;
            noteFlag = true;
            eLog.write('Info', 'Added CMDB CI to work notes.');
        }

        if (source.u_close_notes != '') {
            if (EXECUTION == 'reflect') {
                incident.close_notes = source.u_close_notes;
                incidentFlag = true;
                eLog.write('Info', 'Set close notes.');
            } else {
                notes = notes + '\nClose notes: ' + source.u_close_notes;
                noteFlag = true;
                eLog.write('Info', 'Added close notes to work notes.');
            }
        }

        if (source.u_close_code != '') {
            if (EXECUTION == 'reflect') {
                incident.close_code = CLOSE_CODE;
                incidentFlag = true;
                eLog.write('Info', 'Set close code.');
            } else {
                notes = notes + '\nClose code: ' + CLOSE_CODE;
                noteFlag = true;
                eLog.write('Info', 'Added close code to work notes.');
            }
        }

        if (source.u_category != '') {
            if (EXECUTION == 'reflect') {
                incident.category = CATEGORY;
                incidentFlag = true;
                eLog.write('Info', 'Set category.');
            } else {
                notes = notes + '\nCategory: ' + CATEGORY;
                noteFlag = true;
                eLog.write('Info', 'Added category to work notes.');
            }
        }

        if (source.u_caller != '') {
            if (EXECUTION == 'reflect') {
                incident.caller = CALLER;
                incidentFlag = true;
                eLog.write('Info', 'Set caller.');
            } else {
                notes = notes + '\nCaller: ' + source.u_caller;
                noteFlag = true;
                eLog.write('Info', 'Added caller to work notes.');
            }
        }

        if (source.u_assignment_group != '') {
            if (EXECUTION == 'reflect') {
                incident.assignment_group = ASSIGNMENT_GROUP;
```

```
                incidentFlag = true;
                eLog.write('Info', 'Set assignment group.');
            } else {
                notes = notes + '\nAssignment group: ' + source.u_assignmer
                noteFlag = true;
                eLog.write('Info', 'Added assignment group to work notes.')
            }
        }

        if (incidentFlag) {
            incident.update();
            eLog.write('Info', 'Updated incident.');
        }

        if (noteFlag) {
            incident['work_notes'].setJournalEntry(notes);
            incident.update();
            eLog.write('Info', 'Updated work notes.\nNotes:\n' + notes);
        }

        // make sure that we only add the company record to 'eBonded with'
        // when company is not listed
        // there is a scenario where the supplier is eBonding an additional
        // to a ticket they are already eBonded with on a different ticket
        var arrUtil = new ArrayUtil();
        var arr = incident.u_ebonded_with.toString().split(',');
        var pos = arrUtil.indexOf(arr, COMPANY);
        if (pos < 0) {
            arr.push(COMPANY);
            incident.u_ebonded_with = arr.toString();
            incident.update();
            eLog.write('Info', 'Added company record ' + COMPANY + ' to "eB
        }

    } else {
        // this is very bad
        // we should never get here
        eLog.write('High', 'Internal Error: Incident not found. Incident:
    }

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

49. Click **Submit**.

50. In the **Transform Scripts** tab, click **New**.

51. In the **Transform Script New record** section, fill in the following fields:

    - When: onBefore
    - Order: 300
    - Script:

```

```

```
// set number and sys_id
(function runTransformScript(source, map, log, target /*undefined onStart*/

    var eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Transform Maps';
    eLog.u_name = 'eBond Incident Transform';
    eLog.u_source = '[Transform Script] set number and sys_id';
    eLog.u_supplier = SUPPLIER;
    eLog.write('Info', 'Entering.');

    // deBond check
    if (EXECUTION == 'debond') {
        eLog.write('Info', 'Exiting.');
        return;
    }

    source.u_number = INCIDENT_NUMBER;
    source.u_sys_id = INCIDENT;

    eLog.write('Info', 'Exiting.');
    return;
})(source, map, log, target);
```

52. Click **Submit**.
53. Click **Update**.

Data Map - Inbound

The data map table *[u_ebond_data_map]* is the lookup table used to validate field values for inbound messages passed in by a supplier. It is best practice to assume the data the supplier is incorrect and to validate the data the supplier has passed before inserting the data into a ticket.

1. Navigate to **eBond** > **eBond Data Maps**.
2. Click **New**.
    1. In the **eBond Data Map New record** section, fill in the field values listed below.
    2. Click **Submit**.

Module	Classification	Direction	Supplier	Source value	Supplier value	Note
incident	category	inbound	All	Break-fix	Break-fix	
incident	category	inbound	All	Managed Service	Managed Service	
incident	subcategory	inbound	All	Troubleshoot	Troubleshoot	
incident	subcategory	inbound	All	Setup/Configuration	Setup/Configuration	
incident	subcategory	inbound	All	Performance	Performance	
incident	subcategory	inbound	All	Access Control	Access Control	
incident	subcategory	inbound	All	Connectivity	Connectivity	

Module	Classification	Direction	Supplier	Source value	Supplier value	Note
incident	subcategory	inbound	All	Support	Support	
incident	state	inbound	All	1	1	New
incident	state	inbound	All	2	2	In Progress
incident	state	inbound	All	3	3	On Hold
incident	state	inbound	All	6	6	Resolved
incident	state	inbound	All	7	7	Closed
incident	state	inbound	All	8	8	Canceled
incident	impact	inbound	All	1	1	1 - High
incident	impact	inbound	All	2	2	2 - Medium
incident	impact	inbound	All	3	3	3 - Low
incident	urgency	inbound	All	1	1	1 - High
incident	urgency	inbound	All	2	2	2 - Medium
incident	urgency	inbound	All	3	3	3 - Low
incident	hold_reason	inbound	All	1	1	Awaiting Caller
incident	hold_reason	inbound	All	5	5	Awaiting Change
incident	hold_reason	inbound	All	3	3	Awaiting Problem
incident	hold_reason	inbound	All	4	4	Awaiting Vendor
incident	contact_type	inbound	All	email	email	
incident	contact_type	inbound	All	monitoring	monitoring	
incident	contact_type	inbound	All	phone	phone	
incident	contact_type	inbound	All	self-service	self-service	
incident	contact_type	inbound	All	virtual_agent	virtual_agent	
incident	contact_type	inbound	All	walk-in	walk-in	
incident	contact_type	inbound	All	ebond	ebond	

Module	Classification	Direction	Supplier	Source value	Supplier value	Note
incident	close_code	inbound	All	Solved (Work Around)	Solved (Work Around)	
incident	close_code	inbound	All	Solved (Permanently)	Solved (Permanently)	
incident	close_code	inbound	All	Solved Remotely (Work Around)	Solved Remotely (Work Around)	
incident	close_code	inbound	All	Solved Remotely (Permanently)	Solved Remotely (Permanently)	
incident	close_code	inbound	All	Not Solved (Not Reproducible)	Not Solved (Not Reproducible)	
incident	close_code	inbound	All	Not Solved (Too Costly)	Not Solved (Too Costly)	
incident	close_code	inbound	All	Closed/Resolved by Caller	Closed/Resolved by Caller	

Category & Subcategory

It is the author's opinion that the values for *category* and *subcategory* for an incident within ServiceNow is better suited to describe the type of incident reported; rather than the general topic area the incident occurred. Traditionally with ITSM solutions category and subcategory are used to describe the area where an incident occurred. Out of the box, ServiceNow provides the means to relate incident reports directly to the impacted configuration item (CI) within the CMDB; which is a clearer description of the area of interest the incident lays. This frees up the *category* and *subcategory* fields to represent the type of incident being investigated and the being done to resolve the incident. The update set contains these new values for *category* and *subcategory*, but does not remove the out of the box values set by ServiceNow. Each enterprise will have to ascertain which behavior for the fields to use.

Contact Type

By default the transform script *[onStart] Initialize global variables* defaults the contact type to *ebond*. This is not a mandatory field for the supplier, however if not specified the assumption is the ticket originated via an eBond action.

1. Navigate to **Incident** > **All**, click on any incident record.
2. Right-click on **Contact type**, click **Configure Dictionary**.
3. Under the **Choices** tab, click **New**.
4. In the **Choice New record**section, fill in the following fields:
    - Label: eBond

- Value: ebond

5. Click **Submit**.

Multi-source Outbound Handling

Outbound Business Rule

1. Navigate to **System Definition** > **Business Rules**, click **New**.
2. In the **Business Rule New record** section, fill in the following fields:
   - Name: eBond Incident Outbound
   - Table: Incident [incident]
   - Advanced: true
3. In the record header, right-click and select **Save**.
4. In the **When to run** tab, fill in the following fields:
   - Insert: true
   - Update: true
   - Delete: true
5. In the record header, right-click and select **Save**.
6. In the **When to run** tab, fill in the following fields:
   - Condition: new eBondIncident().checkCondition();
   - Script:

```
(function executeRule(current, previous /*null when async*/ ) {

    var dataSet = {};

    if (current.operation == 'insert') {
        dataSet.comment = current.comments.getJournalEntry(1);
        dataSet.work_note = current.work_notes.getJournalEntry(1);
        dataSet.description = current.description.toString();
        dataSet.short_description = current.short_description.toString();
        dataSet.state = current.state.toString();
        dataSet.impact = current.impact.toString();
        dataSet.urgency = current.urgency.toString();
        dataSet.caller = current.caller_id.email.toString();
        dataSet.assignment_group = current.assignment_group.toString();
        dataSet.ci = current.cmdb_ci.toString();
        dataSet.contact_type = current.contact_type.toString();
        dataSet.close_notes = current.close_notes.getDisplayValue();
        dataSet.close_code = current.close_code.toString();
        dataSet.service_offering = current.business_service.toString();
        dataSet.service_offering = current.service_offering.toString();
        dataSet.category = current.category.toString();
        dataSet.subcategory = current.subcategory.toString();
    } else if (current == 'update') {
        if (current.comments.changes()) {
            dataSet.comment = current.comments.getJournalEntry(1);
        }
```

```
        if (current.work_notes.changes()) {
            dataSet.work_note = current.work_notes.getJournalEntry(1);
        }
        if (current.description.changes()) {
            dataSet.description = current.description.toString();
        }
        if (current.short_description.changes()) {
            dataSet.short_description = current.short_description.toString();
        }
        if (current.state.changes()) {
            dataSet.state = current.state.toString();
        }
        if (current.impact.changes()) {
            dataSet.impact = current.impact.toString();
        }
        if (current.urgency.changes()) {
            dataSet.urgency = current.urgency.toString();
        }
        if (current.caller_id.changes()) {
            dataSet.caller = current.caller_id.email.toString();
        }
        if (current.assignment_group.changes()) {
            dataSet.assignment_group = current.assignment_group.toString();
        }
        if (current.cmdb_ci.changes()) {
            dataSet.ci = current.cmdb_ci.toString();
        }
        if (current.contact_type.changes()) {
            dataSet.contact_type = current.contact_type.toString();
        }
        if (current.close_notes.changes()) {
            dataSet.close_notes = current.close_notes.getDisplayValue();
        }
        if (current.close_code.changes()) {
            dataSet.close_code = current.close_code.toString();
        }
        if (current.business_service.changes()) {
            dataSet.service_offering = current.business_service.toString();
        }
        if (current.service_offering.changes()) {
            dataSet.service_offering = current.service_offering.toString();
        }
        if (current.category.changes()) {
            dataSet.category = current.category.toString();
        }
        if (current.subcategory.changes()) {
            dataSet.subcategory = current.subcategory.toString();
        }
    } else { // delete
        // intentionally do nothing
    }

    var dataBundle = JSON.stringify(dataSet);

    gs.eventQueue('ebond.incident.outbound', current, current.operation.toString

})(current, previous);
```

7. Click **Update**.
8. Navigate to **System Policy** > **Events** > **Registry**, click **New**.
9. In the **Event Registration New record**section, fill in the following fields:
    - Name: ebond.incident.outbound
10. Click **Submit**.
11. Navigate to **System Policy** > **Events** > **Script Actions**, click **New**.
12. In the **Script Action New record**section, fill in the following fields:
    - Name: eBond Incident Outbound
    - Event name: ebond.incident.outbound
    - Active: true
    - Script:

```
var operation = event.parm1;
var dataBundle = JSON.parse(event.parm2);
var returnStatus = new eBondIncident().sendIncident(operation, dataBundle);
```

13. Click **Submit**.