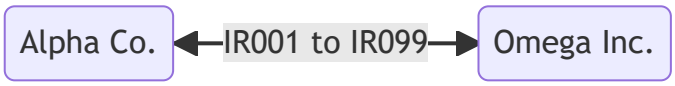
Multi-source eBonding

About

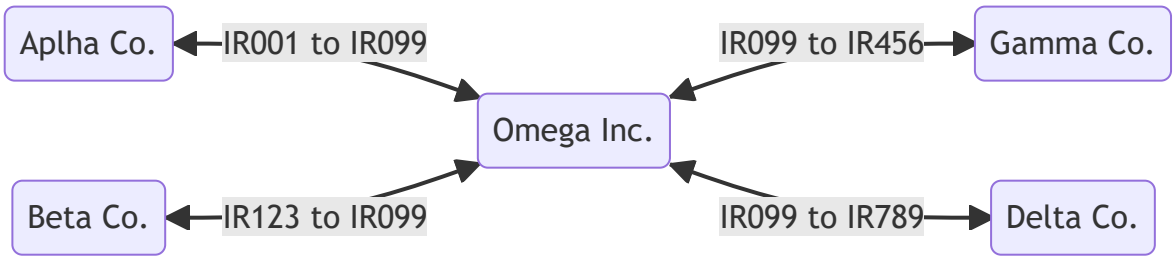
eBonding is a term used to describe the electronic passing of information for business-to-business (B2B) operations. Typically eBonding is used to keep two system of records in sync; e.g., incident reports, service requests, change requests, problem reports, and CMDB configuration items. A record on 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 system of 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: Omega Inc. has contracted with Alpha Co., Beta Co., Gamma Co., and Delta Co. where each supplier provides a distinct set of services. When cooperation between suppliers is required, Omega Inc. coordinates who, how, when, and what information is shared across the suppliers from a single system of record. This allows Omega Inc. to orchestrate multiple suppliers simultaneously to address an issue that spans multiple service domains supported by different suppliers. In this example Omega Inc. has one ticket (IR099) that ties together with other suppliers to address the incident. Each supplier has their own ticket which they can work independently and be kept up to date by other IT service suppliers.



This guide is to help ServiceNow admins to setup multi-source eBonding framework for incident tickets. Incidents are 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, this make incidents the first type of system of record to build a multi-source eBond framework for. The same framework can be applied to other ticket types in ServiceNow with some minor adjustments.

Even if you do not think you will be using multi-source eBonding, this framework performs just as well for direct eBonding and reduce development overhead.

eBonding versus integration

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 developers. 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 system of record, then that party will pass (push) the delta change to the agreed parties.

The difference between eBonding and integration is partnership and cooperation. Where the greater the partnership and cooperation from the supplier yields to an eBond model. The less partnership and cooperation from the supplier yields to an integration model.

Follow along

ServiceNow offers many ways to solve a problem or configure an operational business 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 requirements.

Each section will include the manual instructions setting up the framework and an update set for those that wish to install the multi-source eBonding on their instance. Some of the update sets will have dependencies on other update sets as the functions are shared across functions within the multi-source eBonding.

> **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

This guide 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 and/or development team. This framework is meant to be as open and flexible to meet varying modifications to suit present and future requirements.

Cost Impact

Custom tables are created in this framework and there could be a financial impact creating custom tables. ServiceNow allots a set amount of custom tables within an instance that a customer can create free of charge. After the set amount of custom tables are created, ServiceNow charges for custom tables. Talk with your ServiceNow Enterprise Account Executive to learn more.

Reduce Cost Impact

To reduce any cost impact to your organization, you can extend OOTB tables free of charge. This guide leaves it to the implementation team to determine the best course of action.

Demo data

The guide 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.

Setting a foundation of how things work

The foundation for incident handling 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" in coordination with the assigned to field 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 eBonding 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 within this framework.

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 when removed from the company record.

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.

Instructions:

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 the supplier accounts that will be used by Alpha Co., Beta Co., and Gamma Co. to make RESTful calls for eBonding. Repeat these steps for each supplier. It is suggested that all eBond accounts start with "eBond" to make it easier to filter on.

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**.

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.

Instructions:

1. Navigate to **System Definition** > **Tables** and open the *core_company* record.
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
    - Column name: (this should default to u_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
    - Column name: (this should default to u_ebonded)
8. Under the **Default value** tab, fill in the following field:
    - Default value: false
9. Click **Submit**.

Optional Instructions: 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** > **View** > **Default view**.
3. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Company form**.
4. Under the **Available** list, find the *eBond account* and *eBonded* fields and add them **>** to the **Selected** list.

5. Move the *eBond account* and *eBonded* fields up the **Selected** list ^ to be below the *Stock price*.
6. Click **Save**.

Optional Instructions: 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 field in company record.
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 the company records that will represent Alpha Co., Beta Co., and Gamma Co. to make RESTful calls for eBonding. Repeat these steps for each supplier.
>
> 1. Navigate to **Organization** > **Companies**, click **New**.
> 2. Under the **Company New Record** section, fill in the following field:
>   - Name: Alpha Co.
>   - Stock symbol: AlphaCo
>   - eBonded: True
>   - eBond account: Alpha Co eBond Account
> 3. Click **Submit**.

Associate suppliers with groups

Associating a supplier with an assignment group helps automation know to whom the ticket should be eBonded with. All IT service management tickets (Service Requests, Change Requests, Incidents, Problem Reports) extend the [task] table. The Task table contains two fields of note; Service and Service Offering. The Service narrows down which Service Offerings to choose from and the Service Offering has a field of what group supports the offering. The group could be internal or external to your organization. If external, in the case of most eBonds, then you want to associate that support group with the supplier's company record.

Instructions:

1. Navigate to **System Definition** > **Tables** and open the *[sys_user_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
    - Column name: (this should default to u_company)
4. Under the **Reference Specification** tab, fill in the following fields:
    - Table to reference: Company [core_company]
5. Click **Submit**.
6. Click **Update**.

Optional Instructions: 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** > **View** > **Default view**.
3. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Group form**.
4. Under the **Available** list, find the *Company* field and add it **>** to the **Selected** list.
5. Move the *Company* field up the **Selected** list **^** to be below the *Name*.
6. 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: Call Center Operations
    - Company: Alpha Co.
3. Click **Submit**.
4. Click **New**.
5. Under the **Group New Record** section, fill in the following fields:
    - Name: Data Center Support
    - Company: Beta Co.
6. Click **New**.
7. Under the **Group New Record** section, fill in the following fields:
    - Name: Backup and Recovery Support
    - Company: Gamma Co.
8. Click **Submit**.

Service, service offering, and assignment group alignment in incidents

Ticket mis-routing is problem in all services organizations and surveys have shown negative user experience impacts 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.

Instructions:

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 **Advanced** 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	Call Center Operations
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
    - Support Group: Call Center Operations
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

ServiceNow fulfillers (typically accounts with the itil role) will need the ability to eBond incidents with suppliers. 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."

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

> **eBonded with**
>
> The custom field *eBonded with* 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. You will have to work with the supplier on if and how re-eBonding works.

Instructions:

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** > **View** > **Default view**.
8. Navigate to **Additional actions** > **Configure** > **Form Layout**, which will bring up the **Configuring Company form**.
9. Under the **Available** list, find the *eBonded with* field and add it **>** to the **Selected** list.
10. Move the *eBonded with* field up the **Selected** list **^** to be below the *Assigned to*.
11. 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 a ticket to be mapped to multiple supplier tickets. The fields relate the ServiceNow ticket to the various suppliers the ticket is eBonded too.

The [u_eBond_relationship] table will contain the following custom fields:

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

Instructions:

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: Reference

    - Column label: Company
    - Column name: (this should default to u_company)
7. 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
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: Correlation ID
    - Column name: (this should default to u_correlation_id)
    - Max length: 32
11. Click **Submit**.
12. In the **Columns** table, click **New**.
13. 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
14. Click **Submit**.
15. In the **Columns** table, click **New**.
16. 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)
17. In the **Choice List Specification** tab, select **Dropdown with --None--**.
18. In the record header, right-click and select **Save**.
19. 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
300	Wait	wait

20. Click **Update**.
21. In the **Columns** table, click **New**.
22. 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)
23. In the **Choice List Specification** tab, select **Dropdown with --None--**.

24. In the record header, right-click and select **Save**.
25. 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
300	Wait	wait

26. Click **Update**.
27. In the **Columns** table, click **New**.
28. 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)
29. Under the **Default Value** tab, , fill in the following field:
    - Default value: false
30. Click **Submit**.
31. In the **Columns** table, click **New**.
32. 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)
33. In the **Choice List Specification** tab, select **Dropdown with --None--**.
34. In the record header, right-click and select **Save**.
35. 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	New	1
200	In Progress	2
300	On Hold	3
400	Resolved	6
500	Closed	7
600	Canceled	8

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

39. In the **Choice List Specification** tab, select **Dropdown with --None--**.
40. In the record header, right-click and select **Save**.
41. 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

42. Click **Update**.
43. In the **Columns** table, click **New**.
44. 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)
45. Click **Submit**.
46. In the **Columns** table, click **New**.
47. 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)
48. Under **Related Links**, click **Advanced view**.
49. Under the **Dependent Field** tab, click **Use dependent field**.
50. In the **Dependent on field**, select **Source table**.
51. Click **Update**.
52. In the **Columns** table, click **New**.
53. 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
54. Click **Submit**.
55. Click **Update**.

Relationship Support Script

Helper function that updates the u_ebond_relationship fields.

Instructions:

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)
- Description: Utility function that updates the u_ebond_relationship fields.
- 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.

Instructions:

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: In
    - Value: wait
    - Style: background-color:DarkGoldenrod;color:DarkGoldenrod
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: up
    - Style: background-color:green;color:green
12. Click **Submit**.
13. Click **New**.
14. 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
15. Click **Submit**.
16. In the **Style New record** section, fill in the following fields:
    - Table: u_ebond_relationship
    - Field name: Out
    - Value: wait
    - Style: background-color:DarkGoldenrod;color:DarkGoldenrod
17. Click **Submit**.

Relating eBonded tickets

ServiceNow fulfillers (typically accounts with the itil role) will want to see what supplier tickets are eBonded with the incident ticket they have open. Here we will create a related list that will show the supplier tickets eBonded to an incident.

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

Instructions:

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 to reduce custom table creation, then make sure to look through the various scripts that reference the [u_ebond_registry] table in this guide and change as appropriate.

The [u_ebond_registry] table will contain the following custom fields:

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

Instructions:

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: Key
   - Column name: (this should default to u_key)

    - Max length: 4000
7. Click **Submit**.
8. 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
9. Click **Submit**.
10. In the **Columns** table, click **New**.
11. 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
12. Under the **Default Value** tab, fill in the following field:
    - Default value: All
13. Click **Submit**.
14. 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
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 in this guide that reference the [u_ebond_log] table and change accordingly. The [syslog] table has a granularity of seconds and ServiceNow performs multiple operations per second. Hence the order of the log messages are not guaranteed to be in order. It is recommended that you use the [u_ebond_log] table 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. This needs to be setup first because this will be referenced later on when creating the [u_ebond_log] table.

Instructions:

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)
    - Description: Utility function that updates the u_ebond_log fields.
    - Script:

```
var eBondLog = Class.create();
eBondLog.prototype = {
    initialize: function() {
        this.u_direction = 'not set';
        this.u_location = 'not set';
        this.u_name = 'not set';
        this.u_source = 'not set';
        this.u_supplier = 'not set';
        this.u_message = 'not set';
        this.u_level = 500; // debug
        this.u_correlate_id = 'not set';
        this.u_correlate_class_name = 'not set';

        var reg = new GlideRecord('u_ebond_registry');
        reg.addQuery('u_supplier', 'All');
        reg.addQuery('u_key', 'inbound.log.level');
        reg.query();
        if (reg.next()) {
            this.logLevel = parseInt(reg.value);
        } else {
            this.logLevel = 1000;
        }
    },

    // 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
    },

    // converts the string level to numeric value
    pitch: function(level) {
        var eval = level.toLowerCase();
        var val = 1000;
        switch(eval) {
            case 'high':
                val = 100;
                break;
```

```
            case 'medium':
                val = 200;
                break;
            case 'low':
                val = 300;
                break;
            case 'info':
                val = 400;
                break;
            case 'debug':
                val = 500;
                break;
            default:
                val = 1000;
        }
        return val;
    },

    // records the log message
    write: function(level, message) {
        // only log levels at or below what has been registered
        if (this.logLevel < this.pitch(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.u_correlate_id = this.u_correlate_id;
        log.u_correlate_class_name = this.u_correlate_class_name;
        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*.

Instructions:

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();
- Field type: Integer
- Available for default: true

4. Click **Submit**.

---

eBond Log

The [u_ebond_log] table will contain the following custom fields:

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

Instructions:

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: String
   - Column label: Correlate ID
   - Column name: (this should default to u_correlate_id)
   - 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: Correlate class name
- Column name: (this should default to u_correlate_class_name)
- Max length: 100

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)
13. In the **Choice List Specification** tab, select **Dropdown with --None--**.
14. In the record header, right-click and select **Save**.
15. 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

16. Click **Update**.
17. In the **Columns** table, click **New**.
18. 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
19. Under **Related Links**, click **Advanced view**.
20. Under the **Default Value** tab, fill in the following field:
    - Use dynamic default: true
    - Dynamic Default value: eBond Log Index Increment
21. Click **Update**.
22. In the **Columns** table, click **New**.
23. 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)
24. In the **Choice List Specification** tab, select **Dropdown with --None--**.
25. In the record header, right-click and select **Save**.
26. 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

Sequence	Label	Value
100	High	100
200	Medium	200
300	Low	300
400	Info	400
500	Debug	500

27. Click **Update**.
28. In the **Columns** table, click **New**.
29. 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
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: Message
    - Column name: (this should default to u_message)
    - Max length: 4000
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: Name
    - Column name: (this should default to u_name)
    - 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: Source
    - Column name: (this should default to u_source)
    - Max length: 80
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: Supplier
    - Column name: (this should default to u_supplier)
    - Max length: 40
42. Click **Submit**.
43. Click **Update**.

Optional Instructions: Create the [u_ebond_registry] record used to determine the depth of logs levels to record; else, by default no eBond logs will be written to the [u_ebond_log] table.

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**.

---

eBond Data Map

The [u_ebond_data_map] table is a translation data table for inbound and outbound communication with suppliers. Use this table to map and verify incoming and outgoing data with suppliers.

The [u_ebond_data_map] table will contain the following custom fields:

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

Instructions:

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: Classification
    - Column name: (this should default to u_classification)
    - 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: Choice
- Column label: Direction
- Column name: (this should default to u_direction)

10. In the **Choice List Specification** tab, select **Dropdown with --None--**.
11. In the record header, right-click and select **Save**.
12. 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
300	Duplex	duplex

13. Click **Update**.
14. In the **Columns** table, click **New**.
15. 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
16. Click **Submit**.
17. 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
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
    - Column name: (this should default to u_supplier)
    - Max length: 40
24. Click **Submit**.
25. In the **Columns** table, click **New**.
26. 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

27. Click **Submit**.
28. Click **Update**.

---

eBond Data Map Script Include

Utility script used to retrieve data from the [u_ebond_data_map] table.

Instructions:

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.u_correlate_id = current.sys_id;
        this.eLog.u_correlate_class_name = current.sys_class_name;
        this.eLog.write('Debug', 'Entering.');

        this.eLog.write('Debug', '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('Debug', 'Entering.');
        this.eLog.write('Debug', 'Parms:\n supplier = ' + supplier + '\ndirectic

        dataMap = {};
```

```
        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()) {
            dataMap.value = supplierValue.u_supplier_value.toString();
            if (supplierValue.u_note != undefined && supplierValue.u_note != '')
                dataMap.note = supplierValue.u_note.toString();
            }
        } else {
            dataMap.value = default_value;
        }

        var dataMapStr = JSON.stringify(dataMap);
        this.eLog.write('Debug', 'Exiting. Return: ' + dataMapStr);
        return dataMap;
    },

    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**.

REST Payload Script Include

Component Type: Standard

The eBondRestPayload script stores and sends REST payloads to suppliers. There are several things that happen within this script. All payloads are stored in the [u_ebond_rest_payload] table. If there is not an existing REST message for the supplier for that specific ticket, then the script will make an async RESTful execution call to the supplier; else, the payload will be stored and processed at a later time.

Instructions:

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

```
var eBondRestPayload = Class.create();
eBondRestPayload.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 = 'eBondRestPayload';
        this.eLog.u_source = 'initialize';
        this.eLog.u_supplier = 'Unknown';
        this.eLog.u_correlate_id = current.sys_id;
        this.eLog.u_correlate_class_name = current.sys_class_name;
        this.eLog.write('Debug', 'Entering.');

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

    // func: increment
    // desc: returns the next index number for u_ebond_rest_payload table
    // parm: n/a
    // retn: index interger vale
    increment: function () {
        var gr = new GlideRecord('u_ebond_rest_payload');
        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
    },

    // func: load
    // desc: loads the REST payload into u_ebond_rest_payload table to be proces
    // parm: company - company record
    //       source_table - the ServiceNow table
    //       source - the ticket sys_id
    //       business_rule - the business rule to process the response
    //       endpoint - url destination for the message
    //       payload - JSON body
    //       format - payload format; JSON, XML, etc.
    //       restMsg - rest message record
    //       httpMethod - rest methos to use
    // retn: n/a
    load: function (company, source_table, source, business_rule, endpoint, payl
        this.eLog.u_source = 'load';
        this.eLog.write('Debug', 'Entering.');

        // open the company record
        var supplier = new GlideRecord('core_company');
        supplier.get(company);

        // add to u_ebond_rest_payload
        var payloadRec = new GlideRecord('u_ebond_rest_payload');
        payloadRec.initialize();
        payloadRec.u_active = true;
```

```
        payloadRec.u_supplier = company;
        payloadRec.u_source = source;
        payloadRec.u_source_table = source_table;
        payloadRec.u_endpoint = endpoint;
        payloadRec.u_payload = payload;
        payloadRec.u_format = format;
        payloadRec.u_rest_message = restMsg;
        payloadRec.u_http_method = httpMethod;
        payloadRec.u_ecc_topic = 'eBond_' + supplier.stock_symbol + '_Response';
        payloadRec.u_status = 'queued';
        payloadRec.insert();
        this.eLog.write('Debug', 'Uploaded payload.');

        // prevent collisions with active REST requests on same ticket with supp
        var payloadChk = new GlideRecord('u_ebond_rest_payload');
        payloadChk.addQuery('u_active', true);
        payloadChk.addQuery('u_supplier', company);
        payloadChk.addQuery('u_source', source);
        payloadChk.addQuery('u_source_table', source_table);
        payloadChk.orderByDesc('u_index');
        payloadChk.query();
        if (payloadChk.next() && payloadChk.getRowCount() > 1) {
            this.eLog.write('Debug', 'Collision detected.');
            this.eLog.write('Debug', 'Exiting.');
            return;
        }

        // send over to the ecc queue
        try {
            var restObj = new sn_ws.RESTMessageV2(restMsg, httpMethod);
            restObj.setStringParameter('endpoint', endpoint);
            restObj.setRequestBody(payload);
            restObj.setEccParameter('skip_sensor', 'true'); // prevent Discovery
            restObj.setEccTopic('eBond_' + supplier.stock_symbol + '_Response');
            restObj.setEccCorrelator(payloadRec.getValue('sys_id'));
            this.eLog.write('Debug', 'Invoking async RESTful execution.');
            restObj.executeAsync(); // a business rule on the ecc_queue table wi
            payloadRec.u_status = 'active';
            payloadRec.update();
            return true;
        } catch (ex) {
            var message = ex.getMessage();
            this.eLog.write('Medium', 'Error (Caught Exception): ' + message);
        }
        this.eLog.write('Debug', 'Exiting.');
        return;
    },

    // func: executeNext
    // desc: executes the next REST call from the u_ebond_rest_payload
    // parm: source - the relationship record
    // retn: n/a
    executeNext: function (source) {
        this.eLog.u_source = 'executeNext';
        this.eLog.write('Debug', 'Entering.');
        this.eLog.write('Debug', 'source: ' + source);
```

```
        var nextPayload = new GlideRecord('u_ebond_rest_payload');
        nextPayload.setLimit(1);
        nextPayload.orderBy('u_index');
        nextPayload.addQuery('u_active', true);
        nextPayload.addNullQuery('u_http_status_code');
        nextPayload.addQuery('u_source', source);
        nextPayload.query();
        if (nextPayload.next()) {
            // send over to the ecc queue
            try {
                var restObj = new sn_ws.RESTMessageV2(nextPayload.u_rest_message
                restObj.setStringParameter('endpoint', nextPayload.u_endpoint);
                restObj.setRequestBody(nextPayload.u_payload);
                restObj.setEccParameter('skip_sensor', 'true'); // prevent Disco
                restObj.setEccTopic(nextPayload.u_ecc_topic);
                restObj.setEccCorrelator(nextPayload.getValue('sys_id'));
                this.eLog.write('Debug', 'Invoking async RESTful execution.');
                restObj.executeAsync(); // a business rule on the ecc_queue tabl
                nextPayload.u_status = 'active';
                nextPayload.update();
            } catch (ex) {
                var message = ex.getMessage();
                this.eLog.write('Medium', 'Error (Caught Exception): ' + message
            }
        }

        this.eLog.write('Debug', 'Exiting.');
        return;
    },

    type: 'eBondRestPayload'
};
```

3. Click **Submit**.

---

eBond REST Payload Index Dynamic Filter Options

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*.

Instructions:

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 eBondRestPayload().increment();
   - Field type: Integer
   - Available for default: true
4. Click **Submit**.

---

eBond REST Payloads

The [u_ebond_rest_payload] table will contain the following custom fields:

Field	Description
Active	True or False
ECC topic	Reference to the relationship table.
Endpoint	The URL endpoint for the REST call.
Format	The format of the payload; e.g., JSON, XML, SOAP, etc. etc..
History	Historical retry information.
HTTP Method	ServiceNow REST method label.
HTTP Response	Entire HTTP response payload.
HTTP Status	HTTP status description.
HTTP Status Code	HTTP status code.
Index	Auto-incrementing numeric value.
Payload	REST data payload for the supplier.
REST Message	ServiceNow REST message label.
Retry cap	The maximum number of retries executing the RESTful call to the supplier.
Retry count	The number of retries performed executing the RESTful call to the supplier.
Source	The ServiceNow record sys_id on the instance.
Source table	The table the ServiceNow record resides on.
Status	The current operational status of the REST call.
Supplier	Reference to the supplier.

Instructions:

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: True/False
    - Column label: Active
    - Column name: (this should default to u_active)
7. Under **Related Links**, click **Advanced view**.

8. Under the **Calculated Value** tab, fill in the following fields:
    - Calculated: true
    - Calculation:

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

    if (current.u_retry_count > current.u_retry_cap) {
        calc = false;
    } else if (current.u_http_status_code == 401 ||
        current.u_http_status_code == 404 ||
        current.u_http_status_code == 408 ||
        (current.u_http_status_code >= 500 &&
            current.u_http_status_code <= 599)) {
        calc = true;
    } else {
        calc = false;
    }

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

9. Click **Update**.
10. In the **Columns** table, click **New**.
11. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: String
    - Column label: ECC topic
    - Column name: (this should default to u_ecc_topic)
    - Max length: 40
12. Click **Submit**.
13. In the **Columns** table, click **New**.
14. 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
15. Click **Submit**.
16. In the **Columns** table, click **New**.
17. 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
18. Click **Submit**.
19. In the **Columns** table, click **New**.
20. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Journal Input
    - Column label: History

    - Column name: (this should default to u_history)
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: HTTP Method
    - Column name: (this should default to u_http_method)
    - Max length: 40
24. Click **Submit**.
25. In the **Columns** table, click **New**.
26. 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
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: HTTP Status
    - Column name: (this should default to u_http_status)
    - Max length: 100
30. Under **Related Links**, click **Advanced view**.
31. Under the **Calculated Value** tab, fill in the following fields:
    - Calculated: true
    - Calculation:

```
(function calculatedFieldValue(current) {

    var response = '';

    if (current.u_http_status_code == undefined || current.u_http_status_co
        return response;
    }

    var httpStatusCode = current.u_http_status_code.toString();
    var key = httpStatusCode.substring(0,1);
    switch (key) {
        case '1':
            if (httpStatusCode == '100') {
                response = 'Continue';
            } else if (httpStatusCode == '101') {
                response = 'Switching Protocols';
            } else if (httpStatusCode == '102') {
                response = 'Processing';
            } else if (httpStatusCode == '103') {
                response = 'Early Hints';
            } else {
                response = 'Unassigned';
            }
            break;
```

```
        case '2':
            if (httpStatusCode == '200') {
                response = 'OK';
            } else if (httpStatusCode == '201') {
                response = ' Created';
            } else if (httpStatusCode == '202') {
                response = 'Accepted';
            } else if (httpStatusCode == '203') {
                response = 'Non-Authoritative Information';
            } else if (httpStatusCode == '204') {
                response = 'No Content';
            } else if (httpStatusCode == '205') {
                response = 'Reset Content';
            } else if (httpStatusCode == '206') {
                response = 'Partial Content';
            } else if (httpStatusCode == '207') {
                response = 'Multi-Status';
            } else if (httpStatusCode == '208') {
                response = 'Already Reported';
            } else if (httpStatusCode == '226') {
                response = 'IM Used';
            } else {
                response = 'Unassigned';
            }
            break;
        case '3':
            if (httpStatusCode == '300') {
                response = 'Multiple Choices';
            } else if (httpStatusCode == '301') {
                response = 'Moved Permanently';
            } else if (httpStatusCode == '302') {
                response = 'Found';
            } else if (httpStatusCode == '303') {
                response = 'See Other';
            } else if (httpStatusCode == '304') {
                response = 'Not Modified';
            } else if (httpStatusCode == '305') {
                response = 'Use Proxy';
            } else if (httpStatusCode == '306') {
                response = '(Unused)';
            } else if (httpStatusCode == '307') {
                response = 'Temporary Redirect';
            } else if (httpStatusCode == '308') {
                response = 'Permanent Redirect';
            } else {
                response = 'Unassigned';
            }
            break;
        case '4':
            if (httpStatusCode == '400') {
                response = 'Bad Request';
            } else if (httpStatusCode == '401') {
                response = 'Unauthorized';
            } else if (httpStatusCode == '402') {
                response = 'Payment Required';
            } else if (httpStatusCode == '403') {
                response = 'Forbidden';
```

```
            } else if (httpStatusCode == '404') {
                response = 'Not Found';
            } else if (httpStatusCode == '405') {
                response = 'Method Not Allowed';
            } else if (httpStatusCode == '406') {
                response = 'Not Acceptable';
            } else if (httpStatusCode == '407') {
                response = 'Proxy Authentication Required';
            } else if (httpStatusCode == '408') {
                response = 'Request Timeout';
            } else if (httpStatusCode == '409') {
                response = 'Conflict';
            } else if (httpStatusCode == '410') {
                response = 'Gone';
            } else if (httpStatusCode == '411') {
                response = 'Length Required';
            } else if (httpStatusCode == '412') {
                response = 'Precondition Failed';
            } else if (httpStatusCode == '413') {
                response = 'Payload Too Large';
            } else if (httpStatusCode == '414') {
                response = 'URI Too Long';
            } else if (httpStatusCode == '415') {
                response = 'Unsupported Media Type';
            } else if (httpStatusCode == '416') {
                response = 'Range Not Satisfiable';
            } else if (httpStatusCode == '417') {
                response = 'Expectation Failed';
            } else if (httpStatusCode == '421') {
                response = 'Misdirected Request';
            } else if (httpStatusCode == '422') {
                response = 'Unprocessable Entity';
            } else if (httpStatusCode == '423') {
                response = 'Locked';
            } else if (httpStatusCode == '424') {
                response = 'Failed Dependency';
            } else if (httpStatusCode == '425') {
                response = 'Too Early';
            } else if (httpStatusCode == '426') {
                response = 'Upgrade Required';
            } else if (httpStatusCode == '427') {
                response = 'Unassigned';
            } else if (httpStatusCode == '428') {
                response = 'Precondition Required';
            } else if (httpStatusCode == '429') {
                response = 'Too Many Requests';
            } else if (httpStatusCode == '430') {
                response = 'Unassigned';
            } else if (httpStatusCode == '431') {
                response = 'Request Header Fields Too Large';
            } else if (httpStatusCode == '451') {
                response = 'Unavailable For Legal Reasons';
            } else {
                response = 'Unassigned';
            }
            break;
        case '5':
```

```
            if (httpStatusCode == '500') {
                response = 'Internal Server Error';
            } else if (httpStatusCode == '501') {
                response = 'Not Implemented';
            } else if (httpStatusCode == '502') {
                response = 'Bad Gateway';
            } else if (httpStatusCode == '503') {
                response = 'Service Unavailable';
            } else if (httpStatusCode == '504') {
                response = 'Gateway Timeout';
            } else if (httpStatusCode == '505') {
                response = 'HTTP Version Not Supported';
            } else if (httpStatusCode == '506') {
                response = 'Variant Also Negotiates';
            } else if (httpStatusCode == '507') {
                response = 'Insufficient Storage';
            } else if (httpStatusCode == '508') {
                response = 'Loop Detected';
            } else if (httpStatusCode == '509') {
                response = 'Unassigned';
            } else if (httpStatusCode == '510') {
                response = 'Not Extended';
            } else if (httpStatusCode == '511') {
                response = 'Network Authentication Required';
            } else {
                response = 'Unassigned';
            }
            break;
        default:
            response = 'Unknown';
            break;
    }
    return response;  // return the calculated value

})(current);
```

32. Click **Update**.
33. In the **Columns** table, click **New**.
34. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: HTTP Status Code
    - Column name: (this should default to u_http_status_code)
35. Click **Submit**.
36. In the **Columns** table, click **New**.
37. 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
38. Under **Related Links**, click **Advanced view**.
39. Under the **Default Value** tab, fill in the following field:
    - Use dynamic default: true

    - Dynamic Default value: eBond REST Payload Index Increment
40. Click **Update**.
41. In the **Columns** table, click **New**.
42. 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
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: REST Message
    - Column name: (this should default to u_rest_message)
    - Max length: 40
46. Click **Submit**.
47. In the **Columns** table, click **New**.
48. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Integer
    - Column label: Retry cap
    - Column name: (this should default to u_retry_cap)
49. Under the **Default Value** tab, , fill in the following field:
    - Default value: 24
50. Click **Submit**.
51. In the **Columns** table, click **New**.
52. 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)
53. Under the **Default Value** tab, , fill in the following field:
    - Default value: 0
54. Click **Submit**.
55. In the **Columns** table, click **New**.
56. 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)
57. Click **Submit**.
58. In the **Columns** table, click **New**.
59. 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)
60. Under **Related Links**, click **Advanced view**.
61. Under the **Dependent Field** tab, click **Use dependent field**.
62. In the **Dependent on field**, select **Source table**.

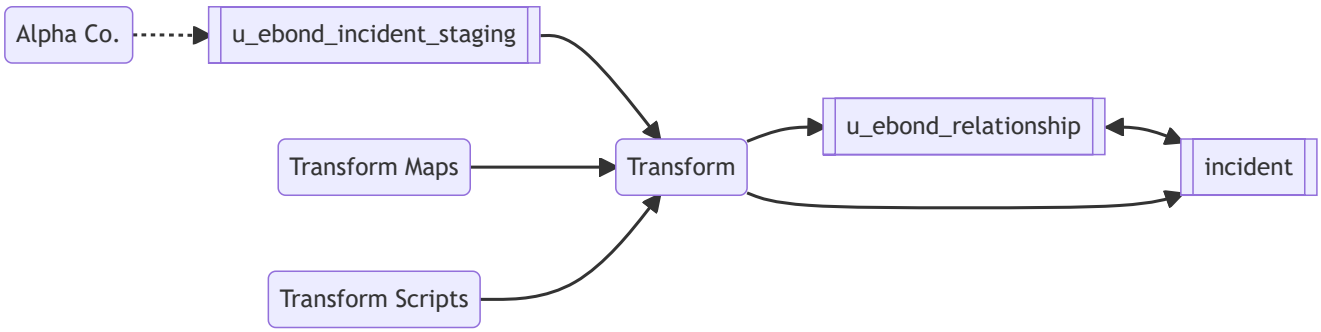
63. Click **Update**.
64. In the **Columns** table, click **New**.
65. In the **Dictionary Entry New record** section, fill in the following fields:
    - Type: Choice
    - Column label: Status
    - Column name: (this should default to u_status)
66. In the record header, right-click and select **Save**.
67. 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	Queued	queued
200	Active	active
300	Processed	processed

68. Click **Update**.
69. In the **Columns** table, click **New**.
70. 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)
71. 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
72. Click **Submit**.
73. Click **Update**.

Multi-source Inbound Handling

If you are employing the push-push model as suggested by the guide for eBonding, then once the inbound handling framework is setup there is no maintenance or further development required. The only exceptions is when you want to add additional functionality to your ServiceNow instance. As you onboard new suppliers, there is no new development work needed for inbound data processing as it will remain the same across the suppliers.



Operations:

1. Alpha Co. initiates a RESTful call sending over a payload to the [u_ebond_incident_staging] table.
2. ServiceNow detects the inserted row and invokes the transform operation.
3. The transform uses its transform maps and scripts to create or update the incident and relationship.
4. The incident and relationship records are linked.

Quick Rundown

Suppliers will write to a [u_ebond_incident_staging] table. 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, this will save you development effort.

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 has been 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 for change request handling.

The [u_ebond_incident_staging] table holds the field values passed by the supplier for incidents and will contain the following custom fields:

Field	Description

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

Instructions:

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: Assignment Group
    - Column name: (this should default to u_assignment_group)
    - 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: Caller
    - Column name: (this should default to u_caller)
    - 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: String
    - Column label: Category
    - Column name: (this should default to u_category)
    - Max length: 80
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: Close Code
    - Column name: (this should default to u_close_code)
    - Max length: 40
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: Close Notes
    - Column name: (this should default to u_close_notes)
    - Max length: 1000
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: CMDB CI
    - Column name: (this should default to u_cmdb_ci)
    - Max length: 40
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: Comment

    - Column name: (this should default to u_comment)
    - Max length: 4000
25. Click **Submit**.
26. In the **Columns** table, click **New**.
27. 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
28. Click **Submit**.
29. In the **Columns** table, click **New**.
30. 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
31. Click **Submit**.
32. In the **Columns** table, click **New**.
33. 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
34. Click **Submit**.
35. In the **Columns** table, click **New**.
36. 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
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: External Reference
    - Column name: (this should default to u_external_reference)
    - 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: Hold Reason
    - Column name: (this should default to u_hold_reason)
    - 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: Impact
    - Column name: (this should default to u_impact)
    - Max length: 40
46. Click **Submit**.
47. In the **Columns** table, click **New**.
48. 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
49. Click **Submit**.
50. In the **Columns** table, click **New**.
51. 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
52. Click **Submit**.
53. In the **Columns** table, click **New**.
54. 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
55. Click **Submit**.
56. In the **Columns** table, click **New**.
57. 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
58. Click **Submit**.
59. In the **Columns** table, click **New**.
60. 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
61. Click **Submit**.
62. In the **Columns** table, click **New**.
63. 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
64. Click **Submit**.
65. In the **Columns** table, click **New**.
66. 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
67. Click **Submit**.
68. 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
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: Work Note
    - Column name: (this should default to u_work_note)
    - Max length: 4000
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.

This transform takes what the supplier has passed and directly targets the [u_ebond_relationship] table and indirectly targets the [incident] table. For a transform to work, ServiceNow needs to know if it will create or update an existing record. Typically with single source eBonding there is no need for a relationship table and you can use the correlation_display or correlation_id fields in [incident] records to find unique records to coalesce on. Because these fields are string fields, makes it a bit difficult for multi-source eBonding to work well. That would require the transform to try to find the right ticket record. It can be done, but then a lot of functionality is lost not using a eBond relationship table. So instead of transforming supplier data to an [incident] record, the transform will map the data to a [u_ebond_relationship] record and update or create an incident depending on a few scenarios.

Instructions:

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:
    - Target field: Company [u_company]
    - Coalesce: True
    - Use source script: True
    - Script:

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

    // 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 === undefined || source.sys_created_by == ""
        return;
    }
```

```
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 || SUPPLIER == null || SUPPLIER == "" ) {
    var SUPPLIER = 'unknown';
}
eLog.u_supplier = SUPPLIER;
eLog.u_correlate_id = source.sys_id;
eLog.u_correlate_class_name = source.sys_class_name;
eLog.write('Debug', 'Entering.');

// 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('Debug', '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('Debug', '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('Debug', 'Exiting.');
        return company.sys_id;
    }

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

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

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

})(source);
```

13. In the record header, right-click and select **Save**.
14. Change the **Choice action** field to **ignore**.
15. Click **Update**.

> **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.

16. In the **Transform Scripts** tab, click **New**.
17. 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('Debug', '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('Debug', 'Exiting.');
```

18. Click **Submit**.
19. In the **Transform Scripts** tab, click **New**.
20. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', '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('Debug', '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('Debug', 'Exiting.');
    return;
})(source, map, log, target);
```

21. Click **Submit**.
22. In the **Transform Scripts** tab, click **New**.
23. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', '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('Debug', '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('Debug', 'COMPANY = ' + COMPANY);
    }

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

24. Click **Submit**.
25. In the **Transform Scripts** tab, click **New**.
26. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', 'Entering.');

    if (action == 'insert') {
        eLog.write('Debug', '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('Debug', 'RELATIONSHIP = ' + RELATIONSHIP);
        if (relationship.u_status == 'debonded') {
            eLog.write('Low', 'Supplier is passing information on a debonde
            error = true;
            error_message = "The incident has been debonded. Updates are no
            source.u_error = error_message;
        }
    } 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('Debug', 'Exiting.');
    return;
})(source, map, log, target);
```

27. Click **Submit**.
28. In the **Transform Scripts** tab, click **New**.
29. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
```

```
    eLog.write('Debug', 'Entering.');

    /*
      scenario combination matrix

      invalid is where logically the transform cannot handle the scenario
      if OPERATION or EXECUTION found to default to invalid will log an err

      action is set by ServiceNow's operation on the u_ebond_relationship t
      if action is insert, then this is a new relationship
      if action is update, an existing relationship with the supplier on th

      OPERATION is wether to create a new incident or update an existing in

      EXECUTION is wether to directly reflect the changes made by the suppl
      inform the ticket of changes made by the supplier, or debond the tick
      action  |  OPERATION  |  EXECUTION  |  SCENARIO
      ------------------------------------------------------------------
      insert  |  create     |  reflect    | new eBond from supplier
      insert  |  create     |  inform     |   invalid - cannot create an in
      insert  |  create     |  debond     |   invalid - cannot create a deb
      insert  |  update     |  reflect    |   invalid - ticket would alread
      insert  |  update     |  inform     | new eBond from supplier to exis
      insert  |  update     |  debond     |   invalid - cannot create a deb
      update  |  create     |  reflect    |   invalid - supplier cannot re-
      update  |  create     |  inform     |   invalid - cannot create an in
      update  |  create     |  debond     |   invalid - cannot create a deb
      update  |  update     |  reflect    | supplier is updating a ticket
      update  |  update     |  inform     | supplier is updating a ticket t
                                          | supplier is re-eBonding to a pr
      update  |  update     |  debond     | supplier is debonding the ticke
    */

    //  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('Debug', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
        eLog.write('Debug', '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('Debug', 'OPERATION = ' + OPERATION + '\nEXECUTION =
            eLog.write('Debug', '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('Debug', '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('Debug', error_mnessage);
                eLog.write('Debug', '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('Debug', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
        eLog.write('Debug', '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('Debug', 'OPERATION = ' + OPERATION + '\nEXECUTION = ' +
    eLog.write('Debug', '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_num
    relationship.query();
    if (relationship.next()) {
        if (relationship.u_reflect == true) {
            OPERATION = 'update';
            EXECUTION = 'reflect';

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

            eLog.write('Debug', 'OPERATION = ' + OPERATION + '\nEXECUTI
            eLog.write('Debug', '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('Debug', 'Exiting.');
    return;

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

30. Click **Submit**.
31. In the **Transform Scripts** tab, click **New**.
32. 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 = SUPPLIER;
    eLog.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', '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('Debug', 'Exiting.');
    return;
})(source, map, log, target);
```

33. Click **Submit**.

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

35. 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.u_correlate_id = source.sys_id;
```

```
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', '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('Debug', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + '\
            eLog.write('Debug', '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('Debug', '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('Debug', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + '\
            eLog.write('Debug', 'Exiting');
            return;
        } else {
            if (source.u_external_number != '') {
                eLog.write('High', 'Could not find external correlation num
            } else {
                eLog.write('High', 'Could not find the external correlation
            }

            error = true;
```

```
            error_message = "eBond relationship not found.";
            source.u_error = error_message;

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

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

36. Click **Submit**.
37. In the **Transform Scripts** tab, click **New**.
38. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', 'Entering.');

    // deBond check
    if (EXECUTION == 'debond') {
        eLog.write('Debug', '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('Debug', 'URL = ' + URL);
    }
```

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

39. Click **Submit**.
40. In the **Transform Scripts** tab, click **New**.
41. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', 'HOLD_REASON = ' + HOLD_REASON + ' (' + HOL
        } 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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', '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('Debug', 'Exiting.');
    return;
})(source, map, log, target);
```

42. Click **Submit**.
43. In the **Transform Scripts** tab, click **New**.
44. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', 'Entering.');

    // eBond check
    if (EXECUTION != 'debond') {
        eLog.write('Debug', '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('Debug', '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('Debug', 'Removed company record ' + COMPANY + ' fro
        }

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

45. Click **Submit**.
46. In the **Transform Scripts** tab, click **New**.
47. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', 'Entering.');
```

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

    // create a new incident record
    if (OPERATION == 'create' && INCIDENT == '') {
        var grInc = new GlideRecord("incident");
        grInc.initialize();
        grInc.setWorkflow(false); // supress business rules
        grInc.insert();
        INCIDENT_NUMBER = grInc.number;
        INCIDENT = grInc.sys_id.toString();
        eLog.write('Debug', 'INCIDENT_NUMBER = ' + INCIDENT_NUMBER + ' INCI
    }

    // 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('Debug', 'Exiting.');
    return;
})(source, map, log, target);
```

48. Click **Submit**.
49. In the **Transform Scripts** tab, click **New**.
50. 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.u_correlate_id = source.sys_id;
```

```
eLog.u_correlate_class_name = source.sys_class_name;
eLog.write('Debug', 'Entering');

if (EXECUTION == 'debond') {
    eLog.write('Debug', '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()) {

    // create a new incident record
    // some of these variables might not have been set from the supplie
    // but are needed for the creation of the ticket
    if (OPERATION == 'create') {
        incident.subcategory = SUBCATEGORY;
        incident.state = STATE;
        incident.service_offering = SERVICE_OFFERING;
        incident.business_service = SERVICE;
        incident.impact = IMPACT;
        incident.urgency = URGENCY;
        incident.contact_type = CONTACT_TYPE;
        incident.category = CATEGORY;
        incident.caller = CALLER;
        incident.assignment_group = ASSIGNMENT_GROUP;
    }

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

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

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

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

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

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

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

```
            }
        }

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

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

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

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

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

```
        if (source.u_cmdb_ci != '') {
            notes = notes + '\n\tCMDB CI: ' + source.u_cmdb_ci;
            noteFlag = true;
            eLog.write('Debug', '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('Debug', 'Set close notes.');
            } else {
                notes = notes + '\nClose notes: ' + source.u_close_notes;
                noteFlag = true;
                eLog.write('Debug', 'Added close notes to work notes.');
            }
        }

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

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

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

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

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

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

        if (noteFlag) {
            incident['work_notes'].setJournalEntry(notes);
            incident.update();
            eLog.write('Debug', '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('Debug', 'Added company record ' + COMPANY + ' to "e
        }

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

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

51. Click **Submit**.

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

53. 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.u_correlate_id = source.sys_id;
    eLog.u_correlate_class_name = source.sys_class_name;
    eLog.write('Debug', 'Entering.');

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

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

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

54. Click **Submit**.
55. Click **Update**.

Data Map - Inbound

The [u_ebond_data_map] table 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 and repeat for each.
    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	

Module	Classification	Direction	Supplier	Source value	Supplier value	Note
incident	subcategory	inbound	All	Access Control	Access Control	
incident	subcategory	inbound	All	Connectivity	Connectivity	
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	

Module	Classification	Direction	Supplier	Source value	Supplier value	Note
incident	contact_type	inbound	All	walk-in	walk-in	
incident	contact_type	inbound	All	ebond	ebond	
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.

1. Navigate to **System Definition** > **Dictionary**.
2. Under the list view search, fill in the the following search fields and hit enter:
    - Table: =incident
    - Column name: category
3. Open the *incident* record.
4. Under the **Choices** tab, click **New**.
5. In the **Choice New record** section, fill in the following fields:
    - Label: Break-fix
    - Value: Break-fix

6. Click **Submit**.
7. Under the **Choices** tab, click **New**.
8. In the **Choice New record** section, fill in the following fields:
    - Label: Managed Service
    - Value: Managed Service
9. Click **Submit**.
10. Click **Update**.
11. Under the list view search, fill in the the following search fields and hit enter:
    - Table: =incident
    - Column name: category
12. Open the *incident* record.
13. Under the **Choices** tab, click **New**.
14. In the **Choice New record** section, fill in the following fields:
    - Label: Troubleshoot
    - Value: Troubleshoot
    - Dependent value: Break-fix
15. Click **Submit**.
16. Under the **Choices** tab, click **New**.
17. In the **Choice New record** section, fill in the following fields:
    - Label: Setup/Configuration
    - Value: Setup/Configuration
    - Dependent value: Break-fix
18. Click **Submit**.
19. Under the **Choices** tab, click **New**.
20. In the **Choice New record** section, fill in the following fields:
    - Label: Performance
    - Value: Performance
    - Dependent value: Break-fix
21. Click **Submit**.
22. Under the **Choices** tab, click **New**.
23. In the **Choice New record** section, fill in the following fields:
    - Label: Access Control
    - Value: Access Control
    - Dependent value: Managed Service
24. Click **Submit**.
25. Under the **Choices** tab, click **New**.
26. In the **Choice New record** section, fill in the following fields:
    - Label: Connectivity
    - Value: Connectivity
    - Dependent value: Managed Service
27. Click **Submit**.
28. Under the **Choices** tab, click **New**.
29. In the **Choice New record** section, fill in the following fields:
    - Label: Support
    - Value: Support
    - Dependent value: Managed Service

30. Click **Submit**.
31. Click **Update**.

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 **System Definition** > **Dictionary**.
2. Under the list view search, fill in the the following search fields and hit enter:
    - Table: =task
    - Column name: contact_type
3. Open the *incident* record.
4. Under the **Choices** tab, click **New**.
5. In the **Choice New record** section, fill in the following fields:
    - Label: eBond
    - Value: ebond
6. Click **Submit**.
7. Click **Update**.

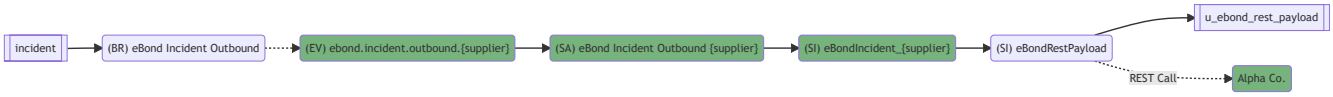
Multi-source Outbound Handling

If you are employing the push-push model as suggested by the guide for eBonding, then there will be a level of development work required for each supplier you onboard. This framework tries to minimize the amount of work required, but because your ServiceNow instance will be "talking" to a supplier's solution you will need some level of customization.

The standard components that make up the outbound framework is similar to that of the inbound framework setup earlier in that no changes need to be made as you onboard suppliers. These components should be static and never change.

The custom components that make up the outbound framework will be unique to each supplier you onboard. Each custom component will have a standard operation that it will need to perform, but how it processes the operation will be unique to each supplier. This guide creates a drop-in framework for suppliers; meaning you should not have to edit existing components to add a new supplier. Rather you add net new components per supplier. This is the same design principles as plugins for software applications.

Each section will inform you if what you are about to setup is a standard component or a custom component at the start of each section. In the diagram below, the custom components are highlighted in green.



Operations:

1. An incident record is updated and a business rule (BR) is alerted of the change.
2. The business rule bundles the data and creates an async event (EV).
3. An event is created and a custom script action (SA) is activated.
4. The script action passes the bundled incident data and passes it to a custom script include (SI) processed.
5. The script include builds out the REST payload and passes it to a standard script include (SI) that stores the payload and sends the payload to the supplier to either create or update a ticket on their side.

Business Rule

Component Type: Standard

When an incident is updated, that update needs to be passed to all of the suppliers that ticket is eBonded with. A Business Rule is the trigger that looks for changes on the table records and then takes action.

This business rule is triggered for any incidents that are or shall be eBonded with one or more suppliers. The business rule gathers all the needed data and then creates a custom event for each supplier that eBonded with the incident. The event handler is a custom component needed for each supplier and will be described below on how those events are handled.

> **When to run**
>
> A question arises why the business rule is set to *after* instead of *async*. It is best practice to free ServiceNow resources to handle operations in an async fashion that way ServiceNow can return control back to the user as soon as possible. This business rule in the end turns the operation into an async operation by passing an async event with the delta changes. What we do not want happening is if this business rule is set to *async*, then there is a possibility of field changes made to the ticket rendering the *changes()* calls incorrect between when the incident is changed and the business rule is executed.

Instructions:

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 **Advanced** tab, fill in the following fields:
    - When: after
    - Insert: true
    - Update: true
    - Delete: true
5. In the record header, right-click and select **Save**.
6. In the **Advanced** tab, fill in the following fields:
    - Condition: new eBondIncident().checkCondition();
    - Script:

```
(function executeRule(current, previous /*null when async*/) {
    var eLog = new eBondLog();
    eLog.u_direction = 'outbound';
    eLog.u_location = 'Business Rules';
    eLog.u_name = 'eBond Incident Outbound';
    eLog.u_source = 'executeRule';
    eLog.u_supplier = 'Unknown';
    eLog.u_correlate_id = current.sys_id;
    eLog.u_correlate_class_name = current.sys_class_name;
    eLog.write('Debug', 'Entering.');

    var dataSet = {};

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

```
            dataSet.state = current.getValue('state');
        }
        if (current.impact.changes()) {
            dataSet.impact = current.getValue('impact');
        }
        if (current.urgency.changes()) {
            dataSet.urgency = current.getValue('urgency');
        }
        if (current.caller_id.changes()) {
            dataSet.caller = current.getValue('caller_id');
        }
        if (current.assignment_group.changes()) {
            dataSet.assignment_group = current.getValue('assignment_group');
        }
        if (current.cmdb_ci.changes()) {
            dataSet.ci = current.getValue('cmdb_ci');
        }
        if (current.contact_type.changes()) {
            dataSet.contact_type = current.getValue('contact_type');
        }
        if (current.close_notes.changes()) {
            dataSet.close_notes = current.getValue('close_notes');
        }
        if (current.close_code.changes()) {
            dataSet.close_code = current.getValue('close_code');
        }
        if (current.business_service.changes()) {
            dataSet.business_service = current.getValue('business_service');
        }
        if (current.service_offering.changes()) {
            dataSet.service_offering = current.getValue('service_offering');
        }
        if (current.category.changes()) {
            dataSet.category = current.getValue('category');
        }
        if (current.subcategory.changes()) {
            dataSet.subcategory = current.getValue('subcategory');
        }
        if (current.u_ebonded_with.changes()) {
            dataSet.ebonded_with = current.u_ebonded_with.toString();
        }
    } else { // delete
        // intentionally do nothing
    }

    var dataBundle = JSON.stringify(dataSet);

    // if nothing has changed in the incident, then go no further
    if (dataBundle == '{}' && current.operation() != 'delete') {
        eLog.write('Debug', 'No reportable changes were made to the incident.');
        eLog.write('Debug', 'Exiting.');
        return;
    }

    // find the suppliers to update
    var suppliers = new eBondIncident().getIncidentSuppliers();
```

```
    // create an event to build out the supplier payload
    for (var index = 0; index < suppliers.length; index++) {
        try {
            gs.eventQueue('ebond.incident.outbound.' + suppliers[index], current
            eLog.write('Debug', 'Queued event ebond.incident.outbound.' + suppli
        } catch (ex) {
            var message = ex.message;
            eLog.write('Medium', 'Caught exception: ' + message);
        }
    }

    eLog.write('Debug', 'Exiting.');

})(current, previous);
```

7. Click **Update**.

Incident Script Include

Component Type: Standard

Utility script that is used by the *Business Rule* setup earlier.

Instructions:

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

```
var eBondIncident = Class.create();
eBondIncident.prototype = {
    // func: initialize
    // desc: initializes the JavaScript object
    // parm: n/a
    // retn: true or false
    initialize: function () {
        this.eLog = new eBondLog();
        this.eLog.u_direction = 'outbound';
        this.eLog.u_location = 'Script Includes';
        this.eLog.u_name = 'eBondIncident';
        this.eLog.u_source = 'initialize';
        this.eLog.u_supplier = 'Unknown';
        this.eLog.u_correlate_id = current.sys_id;
        this.eLog.u_correlate_class_name = current.sys_class_name;
        this.eLog.write('Debug', 'Entering.');

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

    // func: checkIncidentCondition
    // desc: determines if the record is eBonded or should be eBonded
    // parm: n/a
    // retn: true or false
    checkIncidentCondition: function () {
```

```
        this.eLog.u_source = 'checkCondition';
        this.eLog.write('Debug', 'Entering.');

        // check if there are existing eBonded relationships
        var relationships = new GlideRecord('u_ebond_relationship');
        relationships.addQuery('u_source', current.sys_id);
        relationships.addQuery('u_status', 'ebonded');
        relationships.query();
        while (relationships.next()) {
            // check that the company is eBond enabled
            var company = new GlideRecord('core_company');
            company.addQuery('sys_id', relationships.u_company);
            company.addQuery('u_ebonded', 'true');
            company.query();
            if (company.next()) {
                this.eLog.write('Debug', 'Existing relationship(s) found.');
                this.eLog.write('Debug', 'Exiting. Return: true');
                return true;
            }
        }

        // look into the incident
        var incident = new GlideRecord('incident');
        incident.addQuery('sys_id', current.sys_id);
        incident.query();
        if (incident.next()) {
            // check u_ebonded_with
            if (incident.u_ebonded_with != '') {
                this.eLog.write('Debug', 'eBonded with is set.');
                this.eLog.write('Debug', 'Exiting. Return: true');
                return true;
            }
            if (incident.assignment_group) {
                // check assignment group declared in the incident
                var assignmentGroup = new GlideRecord('sys_user_group');
                assignmentGroup.addQuery('sys_id', incident.assignment_group);
                assignmentGroup.query();
                if (assignmentGroup.next()) {
                    // check company associated with the assignment group is eBo
                    var company = new GlideRecord('core_company');
                    company.addQuery('sys_id', assignmentGroup.u_company);
                    company.addQuery('u_ebonded', 'true');
                    company.query();
                    if (company.next()) {
                        this.eLog.write('Debug', 'Incident is assigned to an eBo
                        this.eLog.write('Debug', 'Exiting. Return: true');
                        return true;

                    }
                }
            }
        }

        this.eLog.write('Debug', 'Exiting. Return: false');
        return false;
    },
```

```
    // func: getIncidentSuppliers
    // desc: determines if the record is eBonded or should be eBonded
    // parm: n/a
    // retn: true or false
    getIncidentSuppliers: function () {
        this.eLog.u_source = 'getSuppliers';
        this.eLog.write('Debug', 'Entering.');

        var suppliers = new Array();

        // check if there are existing eBonded relationships
        var relationships = new GlideRecord('u_ebond_relationship');
        relationships.addQuery('u_source', current.sys_id);
        relationships.addQuery('u_status', 'ebonded');
        relationships.query();
        while (relationships.next()) {
            var company = new GlideRecord('core_company');
            company.addQuery('sys_id', relationships.u_company);
            company.addQuery('u_ebonded', 'true');
            company.query();
            if (company.next()) {
                suppliers.push(company.stock_symbol.toString());
            }
        }

        // look at the incident
        var incident = new GlideRecord('incident');
        incident.addQuery('sys_id', current.sys_id);
        incident.query();
        if (incident.next()) {
            // check u_ebonded_with
            if (incident.u_ebonded_with != '') {
                var companies = incident.u_ebonded_with.toString().split(',');
                for (var index = 0; index < companies.length; index++) {
                    var company = new GlideRecord('core_company');
                    company.addQuery('sys_id', companies[index]);
                    company.addQuery('u_ebonded', 'true');
                    company.query();
                    if (company.next()) {
                        suppliers.push(company.stock_symbol.toString());
                    }
                }
            }

            if (incident.assignment_group) {
                // check assignment group declared in the incident
                var assignmentGroup = new GlideRecord('sys_user_group');
                assignmentGroup.addQuery('sys_id', incident.assignment_group);
                assignmentGroup.query();
                if (assignmentGroup.next()) {
                    // check company associated with the assignment group
                    var company = new GlideRecord('core_company');
                    company.addQuery('sys_id', assignmentGroup.u_company);
                    company.addQuery('u_ebonded', 'true');
                    company.query();
                    if (company.next()) {
                        if (company.u_ebonded) {
```

```
                            suppliers.push(company.stock_symbol.toString());
                        }
                    }
                }
            }
        }

        // filter down to unique suppliers
        var arrayUtil = new ArrayUtil();
        var suppliersArray = arrayUtil.unique(suppliers);

        this.eLog.write('Debug', 'Exiting. Returning: ' + suppliersArray.toStrir
        return suppliersArray;
    },

    type: 'eBondIncident'
};
```

3. Click **Submit**.

Event Registry

Component Type: Custom

Each supplier will need it's own registered event within ServiceNow. This is an asynchronous event that will be handled by a **Script Action** that is defined below.

You will need to perform these instructions for AlphaCo, BetaCo, and GammaCo; replacing the supplier stock symbol as you go as the **Business Rule** defined above uses the supplier's stock symbol to generate a portion of the event name.

Instructions:

1. Navigate to **System Policy** > **Events** > **Registry**, click **New**.
2. In the **Event Registration New record** section, fill in the following fields:
    - Event Name: ebond.incident.outbound.
3. Click **Submit**.

Script Actions

Component Type: Custom

Script Actions monitor for registered events being sent. When an event is created and script action associated with the event is activated.

You will need to perform these instructions for AlphaCo, BetaCo, and GammaCo; replacing the supplier stock symbol as you go.

Instructions:

1. Navigate to **System Policy** > **Events** > **Script Actions**, click **New**.
2. In the **Event Registration New record** section, fill in the following fields:
    - Name: eBond Incident Outbound

- Event name: ebond.incident.outbound.
- Script:

```
// build out the data payload for {SUPPLIER}
// store it in u_ebond_rest_payloads to be sent to the {SUPPLIER}
var operation = event.parm1;
var dataBundle = event.parm2;
var returnStatus = new eBondIncident_SUPPLIER().preparePayload(operation, dataBu
```

3. Click **Submit**.

Supplier Script Include

Component Type: Custom

The *Script Action* defined earlier will call the supplier specific script include to bundle up the REST payload to be sent to the supplier. When creating a supplier script include make sure REST payload is passed to the [u_ebond_rest_payload] table.

You will need to perform these instructions for AlphaCo, BetaCo, and GammaCo. Mileage will vary depending how the supplier will want to bundle their data. The example below can act as a good template to build from.

Instructions:

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

```
var eBondIncident_SUPPLIER = Class.create();
eBondIncident_SUPPLIER.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 = 'eBondIncident_SUPPLIER';
        this.eLog.u_source = 'initialize';
        this.eLog.u_supplier = 'SUPPLIER';
        this.eLog.u_correlate_id = current.sys_id;
        this.eLog.u_correlate_class_name = current.sys_class_name;
        this.eLog.write('Debug', 'Entering.');

        this.supplier = 'SUPPLIER';
        this.restMsg = 'SUPPLIER'; // declaration in ServiceNow REST message
        this.httpMethod = 'Incident'; // declaration in ServiceNow REST message

        // dynamic
        var company = new GlideRecord('core_company');
```

```
        company.addQuery('stock_symbol', this.supplier);
        company.query();
        if (company.next()) {
            this.company = company.sys_id;
        } else {
            this.eLog.write('High', 'Cannot find the supplier\'s (' + supplier +
        }

        var url = new GlideRecord('u_ebond_registry');
        url.addQuery('u_supplier', this.supplier);
        url.addQuery('u_key', 'outbound.url.endpoint');
        url.query();
        if (url.next()) {
            this.url = url.u_value;
        } else {
            this.eLog.write('High', supplier + ' is missing eBond registry key c
        }

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

    // func: checkCondition
    // desc: determines if the response is from SUPPLIER; called by ecc queue bu
    // parm: n/a
    // retn: true or false
    checkCondition: function () {
        this.eLog.u_source = 'checkCondition';
        this.eLog.u_direction = 'inbound';
        this.eLog.write('Debug', 'Entering.');

        if (current.getValue('topic') == 'eBond_' + this.supplier + '_Response'
            (current.getValue('state') == 'ready' ||
                current.getValue('state') == 'error') &&
            current.getValue('name') == 'post' &&
            current.getValue('queue') == 'input') {
            this.eLog.write('Debug', 'Exiting. Return: true');
            return true;
        }

        this.eLog.write('Debug', 'Exiting. Return: false');
        return false;
    },

    /*
       func: checkEcho
       desc: checks to see if the incident was updated by the supplier
       parm: n/a
       retn: true - the incident was updated by SUPPLIER
             false - the incident was not updated by SUPPLIER

    */
    checkEcho: function () {
        this.eLog.u_source = 'checkEcho';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        var echo = false;
```

```
        var registry = new GlideRecord('u_ebond_registry');
        registry.addQuery('u_supplier', this.supplier);
        registry.addQuery('u_key', 'inbound.account');
        registry.query();
        if (registry.next()) {
            if (registry.u_value == current.sys_updated_by) {
                echo = true;
            }
        }

        this.eLog.write('Debug', 'Exiting. Return: ' + echo);
        return echo;
    },

    // func: determineOperation
    // desc: updates to incident record does not mean updates to the supplier
    //       the existing incident could have been updated with new assignment g
    // parm: n/a
    // retn: create or update
    determineOperation: function () {
        this.eLog.u_source = 'determineOperation';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        var decision = 'create';

        // check if the incident u_ebonded_with AND the assignment group is stil
        var arrUtil = new ArrayUtil();
        var arr = current.u_ebonded_with.toString().split(',');
        var pos = arrUtil.indexOf(arr, this.company);
        if (pos < 0 && current.assignment_group.u_company != this.company) {
            decision = 'delete';
            this.eLog.write('Debug', 'Exiting. Return: ' + decision);
            return decision;
        }

        // find existing relationship records
        var relationship = new GlideRecord('u_ebond_relationship');
        relationship.addQuery('u_source', current.sys_id);
        relationship.addQuery('u_status', 'ebonded');
        relationship.addQuery('u_company', this.company);
        relationship.query();
        if (relationship.next()) {
            decision = 'update';
        }

        this.eLog.write('Debug', 'Exiting. Return: ' + decision);
        return decision;
    },

    // func: insertOperation
    // desc: ServiceNow has inserted a new record
    // parm: dataBundle
    // retn: n/a
    insertOperation: function (dataBundle) {
        this.eLog.u_source = 'insertOperation';
```

```
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        var parser = new JSONParser();
        var incidentData = parser.parse(dataBundle);

        var dataMap = new eBondDataMap();

        var data = {};
        data.ExternalNumber = current.number.toString();
        data.ExternalRef = current.sys_id.toString();

        for (var key in incidentData) {
            var map = dataMap.getSupplierValue(this.supplier, 'outbound', 'incid
            switch (key) {
                case "comment":
                    data.Comment = map.value;
                    break;
                case "work_note":
                    data.WorkNote = map.value;
                    break;
                case "description":
                    data.Desc = map.value;
                    break;
                case "short_description":
                    data.ShortDesc = map.value;
                    break;
                case "state":
                    data.State = map.value;
                    break;
                case "impact":
                    data.Impact = map.value;
                    break;
                case "urgency":
                    data.Urgency = map.value;
                    break;
                case "caller":
                    var caller = new GlideRecord('sys_user');
                    caller.addQuery('sys_id', map.value);
                    caller.query();
                    if (caller.next()) {
                        data.Caller = caller.getDisplayValue('email');
                    }
                    break;
                case "assignment_group":
                    var group = new GlideRecord('sys_user_group');
                    group.addQuery('sys_id', map.value);
                    group.query();
                    if (group.next()) {
                        data.AssignmentGroup = group.getDisplayValue('name');
                    }
                    break;
                case "ci":
                    var ci = new GlideRecord('cmdb_ci');
                    ci.addQuery('sys_id', 'map.value');
                    ci.query();
                    if (ci.next()) {
```

```
                    data.CI = ci.getDisplayValue('name');
                }
                break;
            case "contact_type":
                data.Contact = map.value;
                break;
            case "close_notes":
                data.CloseNotes = map.value;
                break;
            case "close_code":
                data.CloseCode = map.value;
                break;
            case "business_service":
                var service = new GlideRecord('cmdb_ci_service');
                service.addQuery('sys_id', map.value);
                service.query();
                if (service.next()) {
                    data.Service = service.getDisplayValue('name');
                }
                break;
            case "service_offering":
                var service_offering = new GlideRecord('cmdb_ci_service');
                service_offering.addQuery('sys_id', map.value);
                service_offering.query();
                if (service_offering.next()) {
                    data.ServiceOffering = service_offering.getDisplayValue(
                }
                break;
            case "category":
                data.Category = map.value;
                break;
            case "subcategory":
                data.Subcategory = map.value;
                break;
            default:
                break;
        }
    }

    data.CommentHistory = current.comments.getJournalEntry(-1); // full comm
    data.WorkNoteHistory = current.work_notes.getJournalEntry(-1); // full w

    var payload = JSON.stringify(data);
    this.eLog.write('Debug', 'Payload: \n' + payload);

    // create relationship record
    var relationship = new GlideRecord('u_ebond_relationship');
    relationship.initialize();
    relationship.u_source_table = 'incident';
    relationship.u_source = current.sys_id;
    relationship.u_status = 'ebonded';
    relationship.u_state = current.state;
    relationship.u_out = 'wait';
    relationship.u_company = this.company;
    relationship.insert();

    // load REST payload into the pool to be processed
```

```
        var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relat

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

    // func: createOperation
    // desc: ServiceNow has updated an existing record, but this is a new record
    // parm: dataBundle
    // retn: n/a
    createOperation: function () {
        this.eLog.u_source = 'createOperation';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        var dataMap = new eBondDataMap();

        var data = {};
        data.ExternalNumber = current.number.toString();
        data.ExternalRef = current.sys_id.toString();

        data.Comment = current.comments.getJournalEntry(1);
        data.WorkNote = current.work_notes.getJournalEntry(1);
        data.Desc = current.getValue('description');
        data.ShortDesc = current.getValue('short_description');
        data.State = dataMap.getSupplierValue(this.supplier, 'outbound', 'incide
        data.Impact = dataMap.getSupplierValue(this.supplier, 'outbound', 'incid
        data.Urgency = dataMap.getSupplierValue(this.supplier, 'outbound', 'inci
        data.Caller = current.caller.getDisplayValue('email');
        data.AssignmentGroup = current.getDisplayValue('assignment_group');
        data.CI = current.getDisplayValue('cmdb_ci');
        data.Contact = dataMap.getSupplierValue(this.supplier, 'outbound', 'inci
        data.CloseNotes = current.getValue('close_notes');
        data.CloseCode = dataMap.getSupplierValue(this.supplier, 'outbound', 'ir
        data.Service = current.getDisplayValue('business_service');
        data.ServiceOffering = current.getDisplayValue('service_offering');
        data.Category = current.getDisplayValue('category');
        data.Subcategory = current.getDisplayValue('subcategory');
        data.Comment = current.comments.getJournalEntry(-1); // full comment his
        data.WorkNote = current.work_notes.getJournalEntry(-1); // full work not
        var payload = JSON.stringify(data);
        this.eLog.write('Debug', 'Payload: \n' + payload);

        // create relationship record
        var relationship = new GlideRecord('u_ebond_relationship');
        relationship.initialize();
        relationship.u_source_table = 'incident';
        relationship.u_source = current.sys_id;
        relationship.u_status = 'eBonded';
        relationship.u_state = current.state;
        relationship.u_out = 'wait';
        relationship.u_company = this.company;
        relationship.insert();

        // load REST payload into the pool to be processed
        var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relat

        this.eLog.write('Debug', 'Exiting.');
```

```
    },

    // func: updateOperation
    // desc: ServiceNow has updated the record, update the SUPPLIER records as a
    // parm: dataBundle
    // retn: n/a
    updateOperation: function (dataBundle) {
        this.eLog.u_source = 'updateOperation';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        var parser = new JSONParser();
        var incidentData = parser.parse(dataBundle);

        var dataMap = new eBondDataMap();

        // traverse all related eBonds
        var relationship = new GlideRecord('u_ebond_relationship');
        relationship.addQuery('u_source', current.sys_id);
        relationship.addQuery('u_status', 'ebonded');
        relationship.addQuery('u_company', this.company);
        relationship.query();
        while (relationship.next()) {

            var data = {};
            data.number = relationship.u_correlation_number.toString();

            // if reflect, then perform data mapping
            if (relationship.u_reflect == true) {
                for (var key in incidentData) {
                    var map = dataMap.getSupplierValue(this.supplier, 'outbound'
                    switch (key) {
                        case "comment":
                            data.Comment = map.value;
                            break;
                        case "work_note":
                            data.WorkNote = map.value;
                            break;
                        case "description":
                            data.Desc = map.value;
                            break;
                        case "short_description":
                            data.ShortDesc = map.value;
                            break;
                        case "state":
                            data.State = map.value;
                            break;
                        case "impact":
                            data.Impact = map.value;
                            break;
                        case "urgency":
                            data.Urgency = map.value;
                            break;
                        case "caller":
                            var caller = new GlideRecord('sys_user');
                            caller.addQuery('sys_id', map.value);
                            caller.query();
```

```
                            if (caller.next()) {
                                data.Caller = caller.getDisplayValue('email');
                            }
                            break;
                        case "assignment_group":
                            var group = new GlideRecord('sys_user_group');
                            group.addQuery('sys_id', map.value);
                            group.query();
                            if (group.next()) {
                                data.AssignmentGroup = group.getDisplayValue('na
                            }
                            break;
                        case "ci":
                            var ci = new GlideRecord('cmdb_ci');
                            ci.addQuery('sys_id', map.value);
                            ci.query();
                            if (ci.next()) {
                                data.CI = ci.getDisplayValue('name');
                            }
                            break;
                        case "contact_type":
                            data.Contact = map.value;
                            break;
                        case "close_notes":
                            data.CloseNotes = map.value;
                            break;
                        case "close_code":
                            data.CloseCode = map.value;
                            break;
                        case "business_service":
                            var service = new GlideRecord('cmdb_ci_service');
                            service.addQuery('sys_id', map.value);
                            service.query();
                            if (service.next()) {
                                data.Service = service.getDisplayValue('name');
                            }
                            break;
                        case "service_offering":
                            var service_offering = new GlideRecord('cmdb_ci_serv
                            service_offering.addQuery('sys_id', map.value);
                            service_offering.query();
                            if (service_offering.next()) {
                                data.ServiceOffering = service_offering.getDispl
                            }
                            break;
                        case "category":
                            data.Category = map.value;
                            break;
                        case "subcategory":
                            data.Subcategory = map.value;
                            break;
                        default:
                            break;
                    }
                }
            } else { // else, represent the changes as a comment
                var comment = '';
```

```
                var newline = '';
                for (var key in incidentData) {
                    switch (key) {
                        case 'caller':
                            var caller = new GlideRecord('sys_user');
                            caller.addQuery('sys_id', incidentData[key]);
                            caller.query();
                            if (caller.next()) {
                                comment += newline + key + ': ' + caller.getDisp
                            }
                            break;
                        case 'assignment_group':
                            var group = new GlideRecord('sys_user_group');
                            group.addQuery('sys_id', incidentData[key]);
                            group.query();
                            if (group.next()) {
                                comment += newline + key + ': ' + group.getDispl
                            }
                            break;
                        case 'ci':
                            var ci = new GlideRecord('cmdb_ci');
                            ci.addQuery('sys_id', incidentData[key]);
                            ci.query();
                            if (ci.next()) {
                                comment += newline + key + ': ' + ci.getDisplayV
                            }
                            break;
                        case 'business_service':
                            var service = new GlideRecord('cmdb_ci_service');
                            service.addQuery('sys_id', incidentData[key]);
                            service.query();
                            if (service.next()) {
                                comment += newline + key + ': ' + service.getDis
                            }
                            break;
                        case 'service_offering':
                            var service_offering = new GlideRecord('cmdb_ci_serv
                            service_offering.addQuery('sys_id', incidentData[key
                            service_offering.query();
                            if (service_offering.next()) {
                                comment += newline + key + ': ' + service_offeri
                            }
                            break;
                        default:
                            var map = dataMap.getSupplierValue(this.supplier, 'c
                            comment += newline + key + ': ' + map.value;
                            if (map.note != undefined && map.note != '') {
                                comment += ' (' + map.note + ')';
                            }
                    }
                    newline = '\n';
                }
                data.comment = comment;
            }

            var payload = JSON.stringify(data);
            this.eLog.write('Debug', 'Payload: \n' + payload);
```

```
            // update the relationship record if in good standing
            if (relationship.getValue('u_out') == 'up') {
                relationship.u_out = 'wait';
                relationship.update();
            }

            // load REST payload into the pool to be processed
            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_r
        }

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

    // func: deleteOperation
    // desc: ServiceNow has deleted the record, debond associated SUPPLIER recor
    //       update SUPPLIER to debond from the record
    //       deleted the ServiceNow relationship record
    // parm: n/a
    // retn: n/a
    deleteOperation: function () {
        this.eLog.u_source = 'deleteOperation';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');

        // traverse all related eBonds
        var relationship = new GlideRecord('u_ebond_relationship');
        relationship.addQuery('u_source', current.sys_id);
        relationship.addQuery('u_ebonded', 'true');
        relationship.addQuery('u_company', this.company);
        relationship.query();
        while (relationship.next()) {
            // prepare debond payload
            var data = {};
            data.number = relationship.u_correlation_number.toString();
            data.state = '0';
            var payload = JSON.stringify(data);

            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_r

            // change the relationship
            relationship.u_status = 'debonded';
            relationship.update();
        }

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

    // func: preparePayload
    // desc: prepares the incident payloads for SUPPLIER
    // parm: operation - insert, update, or delete
    //       dataBundle - JSON representating the incident
    // retn: n/a
    preparePayload: function (operation, dataBundle) {
        this.eLog.u_source = 'preparePayload';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');
```

```
        this.eLog.write('Debug', 'operation = ' + operation);
        this.eLog.write('Debug', 'dataBundle = ' + dataBundle);

        if (this.checkEcho() == true) {
            this.eLog.u_source = 'preparePayload';
            this.eLog.write('Debug', 'Exiting.');
            return;
        }

        if (this.company == undefined) {
            this.eLog.write('Low', 'Company is not set, cannot continue.');
            this.eLog.write('Debug', 'Exiting.');
            return;
        }

        if (this.url == undefined) {
            this.eLog.write('Low', 'URL is not set, cannot continue.');
            this.eLog.write('Debug', 'Exiting.');
            return;
        }

        if (operation == 'insert') { // new incident record
            this.insertOperation(dataBundle);
            this.eLog.u_source = 'preparePayload';
        } else if (operation == 'update') { // updated incident record
            var decision = this.determineOperation(dataBundle);
            if (decision == 'create') { // decision is to create new record
                this.createOperation(dataBundle);
                this.eLog.u_source = 'preparePayload';
            } else if (decision == 'update') { // decision is to update existing
                this.updateOperation(dataBundle); // decision is to update recor
                this.eLog.u_source = 'preparePayload';
            } else { // decision is to debond
                this.deleteOperation();
                this.eLog.u_source = 'preparePayload';
            }
        } else if (operation == 'delete') { // deleted incident record
            this.deleteOperation();
            this.eLog.u_source = 'preparePayload';
        }

        this.eLog.write('Debug', 'Exiting.');
        return;
    },

    // func: updateRelationship
    // desc: prepares the incident payloads for SUPPLIER on a change of relatior
    // parm: operation - update
    //       dataBundle - JSON representating the status of the relationship
    // retn: n/a
    updateRelationship: function (operation, dataBundle) {
        this.eLog.u_source = 'updateRelationship';
        this.eLog.u_direction = 'outbound';
        this.eLog.write('Debug', 'Entering.');
        this.eLog.write('Debug', 'operation = ' + operation);
        this.eLog.write('Debug', 'dataBundle = ' + dataBundle);
```

```
        var parser = new JSONParser();
        var relationship = parser.parse(dataBundle);

        if (relationship.status == 'debonded') {
            var data = {};
            data.number = current.u_correlation_number.toString();
            data.state = '0'; // SUPPLIER deBond state code
            var payload = JSON.stringify(data);

            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_r
        } else {
            var data = {};
            data.number = current.u_correlation_number.toString();
            data.state = '-1'; // SUPPLIER re-eBond state code
            var payload = JSON.stringify(data);

            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_r
        }

        this.eLog.write('Debug', 'Exiting.');
        return;
    },

    type: 'eBondIncident_SUPPLIER'
};
```

3. Click **Submit**.

REST Messages

Component Type: Custom (not captured in the eBond - Templates update set)

Each supplier will have a uniquely defined outbound connection definition. This is where credentials for the supplier is set along with unique attributes defined by the supplier.

Instructions:

1. Navigate to **System Web Services** > **Outbound** > **REST MEssage**, click **New**.
2. In the **REST Message New record** section, fill in the following fields:
    - Name: SUPPLIER
    - Endpoint: $
    - Authentication type: (note this is very custom to the supplier)
3. In the record header, right-click and select **Save**.
4. In the **HTTP Methods** tab, click "New".
5. In the **HTTP Method New record** section, fill in the following fields:
    - Name: Incident
    - HTTP method: POST
    - Endpoint: $
6. Click **Submit**.
7. Click **Update**.

REST Message Response Handling

Component Type: Custom

The outbound framework defined earlier sends async RESTful payloads; meaning the script does not wait for a response from the supplier before continuing its operations. This business rule handles the responses returned by the supplier.

Instructions:

1. Navigate to **System Definition** > **Business Rules**, click **New**.
2. In the **Business Rule New record** section, fill in the following fields:
    - Name: eBond_SUPPLIER_Response
    - Table: Queue [ecc_queue]
    - Advanced: True
3. In the **Advanced** tab, fill in the following fields:
    - When: after
    - Insert: True
4. In the **Advanced** tab, fill in the following fields:
    - Condition: new eBondIncident_SUPPLIER().checkCondition();
    - Script:

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

    var supplier = 'SUPPLIER';

    eLog = new eBondLog();
    eLog.u_direction = 'inbound';
    eLog.u_location = 'Business Rules';
    eLog.u_name = 'eBond_' + supplier + '_Response';
    eLog.u_source = 'executeRule';
    eLog.u_supplier = supplier;
    eLog.u_correlate_id = current.sys_id;
    eLog.u_correlate_class_name = current.sys_class_name;
    eLog.write('Debug', 'Entering.');

    new eBondIncident_SUPPLIER().processResponse();

    // security check to make sure the sender is a known source
    var url = new GlideRecord('u_ebond_registry');
    url.addQuery('u_supplier', supplier);
    url.addQuery('u_key', 'outbound.url.endpoint');
    url.query();
    if (url.next()) {
        if (current.source != url.u_value) {
            eLog.write('High', 'Security Alert. Inbound source REST message for
            return;
        }
    } else {
        eLog.write('High', supplier + ' is missing eBond registry key outbound.u
        return;
    }

    eLog.write('Debug', 'ECC Queue sys_id: ' + current.sys_id);
    eLog.write('Debug', 'ECC Queue topic: ' + current.topic);
```

```
    eLog.write('Debug', 'ECC Queue state: ' + current.state);
    eLog.write('Debug', 'ECC Queue payload: ' + current.payload);
    eLog.write('Debug', 'ECC Queue response to: ' + current.response_to);

    // evaluate the response
    var eccREST = new RESTECCResponse(current);
    var responseBody = eccREST.getBody();
    var responseObj = JSON.parse(responseBody);

    // pull the REST pool record from the agent correlator
    var restPayloadRec = new GlideRecord('u_ebond_rest_payload');
    restPayloadRec.get(current.response_to.agent_correlator);
    restPayloadRec.u_http_response = eccREST.getBody();
    var httpStatusCode = eccREST.getStatusCode();
    restPayloadRec.u_http_status_code = parseInt(httpStatusCode);
    restPayloadRec.u_status = 'processed';
    restPayloadRec.update();

    // source: https://www.iana.org/assignments/http-status-codes/http-status-co
    // 1xx: Informational - Request received, continuing process
    // 2xx: Success - The action was successfully received, understood, and acce
    // 3xx: Redirection - Further action must be taken in order to complete the
    // 4xx: Client Error - The request contains bad syntax or cannot be fulfille
    // 5xx: Server Error - The server failed to fulfill an apparently valid requ

    // pull the relationship record from the REST pool record
    var relationship = new GlideRecord('u_ebond_relationship');
    relationship.get(restPayloadRec.u_source);
    eLog.write('Debug', 'Relationship: ' + relationship.sys_id);
    var executeNext = false;
    if (httpStatusCode.substring(0, 1) == '2') {
        relationship.u_out = 'up';
        executeNext = true;
    } else {
        relationship.u_out = 'down';
    }
    relationship.u_correlation_number = responseObj.number;
    relationship.u_correlation_id = responseObj.sys_id;
    relationship.update();

    // process the next REST message for the ticket
    if (executeNext) {
        var next = new eBondRestPayload().executeNext(relationship.sys_id);
    }

    // pull the company record for the supplier
    var company = new GlideRecord('core_company');
    company.addQuery('stock_symbol', supplier);
    company.query();
    company.next();

    // pull the incident record from the relationship record
    var incident = new GlideRecord('incident');
    incident.get(relationship.u_source);

    // update the incident u_ebonded_with
    // traverse all related eBonds
```

```
    var relationshipChk = new GlideRecord('u_ebond_relationship');
    relationshipChk.addQuery('u_source', relationship.u_source);
    relationshipChk.addQuery('u_status', 'ebonded');
    relationshipChk.addQuery('u_company', company.sys_id);
    relationshipChk.query();

    var arrUtil = new ArrayUtil();
    var arr = incident.u_ebonded_with.toString().split(',');
    var pos = arrUtil.indexOf(arr, company.getValue('sys_id'));
    if (relationshipChk.getRowCount() > 0 ) {
        if (pos < 0) {
            arr.push(company.getValue('sys_id'));
            incident.u_ebonded_with = arr.toString();
        }
    } else {
        if (pos >= 0) {
            arr.splice(pos,1);
            incident.u_ebonded_with = arr.toString();
        }
    }
    incident.setWorkflow(false);
    incident.update();

    // update the ecc queue record
    current.state = 'processed';
    var gdt = new GlideDateTime();
    current.processed = gdt;
    current.setWorkflow(false);
    current.update();

    eLog.write('Debug', 'Exiting.');
})(current, previous);
```

5. Click **Submit**.

Retry Operations

Component Type: Standard

All outbound REST messages are stored within the [u_ebond_rest_payload] table. There needs to be a scheduled job that looks for recoverable failures and retry again. This guide recommends retrying once an hour for 24 hours. This is an area for future enhancement for alerting ServiceNow administration on failures.

Instructions:

1. Navigate to **System Definition** > **Scheduled Jobs**, click **New**.
2. In the **Automation Creator** section, click **Automatically run a script of your choosing**.
3. In the **Scheduled Job New record** section, fill in the following fields:
    - Name: eBond REST Payload Monitor
    - Run: Periodically
    - Repeat Interval: Days 0 Hours 01 00 00
    - Starting: 2021-01-01 01:00:00
    - Script:

```
this.eLog = new eBondLog();
this.eLog.u_direction = 'outbound';
this.eLog.u_location = 'Scheduled Jobs';
this.eLog.u_name = 'eBond REST Payload Monitor';
this.eLog.write('Debug', 'Entering.');

var restPayloads = new GlideRecord('u_ebond_rest_payload');
restPayloads.addQuery('u_active',true);
restPayloads.orderBy('u_index');
restPayloads.addNotNullQuery('u_http_status_code');
restPayloads.query();

while(restPayloads.next()) {
    if (restPayloads.u_retry_count < restPayloads.u_retry_cap) {
        // update the rest payload
        restPayloads.u_retry_count = restPayloads.u_retry_count + 1;
        restPayloads.u_history = 'HTTP Status Code: ' + restPayloads.u_http_stat
        restPayloads.u_http_status_code = '';
        restPayloads.u_http_status = '';
        restPayloads.http_response = '';
        restPayloads.update();

        // re-try sending over to the ecc queue
        try {
            var restObj = new sn_ws.RESTMessageV2(restPayloads.u_rest_message, r
            restObj.setStringParameter('endpoint', restPayloads.u_endpoint);
            restObj.setRequestBody(restPayloads.u_payload);
            restObj.setEccParameter('skip_sensor', 'true'); // prevent Discovery
            restObj.setEccTopic(restPayloads.u_ecc_topic);
            restObj.setEccCorrelator(restPayloads.getValue('sys_id'));
            restObj.executeAsync(); // a business rule on the ecc_queue table wi
        } catch (ex) {
            var message = ex.getMessage();
            this.eLog.write('Medium', 'Error (Caught Exception): ' + message);
        }
    } else { // out of retries
        restPayloads.u_active = false;
        restPayloads.update();

        // de-activate pending active payloads for the ticket
        var remainingPayloads = new GlideRecord('u_ebond_rest_payload');
        remainingPayloads.addQuery('u_source', restPayloads.u_source);
        remainingPayloads.orderBy('u_active', true);
        remainingPayloads.query();
        while (remainingPayloads.next()) {
            remainingPayloads.u_http_status_code = -1;
            remainingPayloads.u_status = 'processed';
            remainingPayloads.update();
        }
    }
}

this.eLog.write('Debug', 'Exiting.');
```

4. Click **Submit**.

Relationship Handling

Component Type: Standard

If a fulfiller chooses to they can deBond and re-eBond suppliers from the [u_ebond_relationship] table. The table is shown as a related list in the incident and a fulfiller can change the value of the Status field. Changing the Status field should send an update to the supplier to the change in eBond status.

Instructions:

1. Navigate to **System Definition** > **Business Rules**, click **New**.
2. In the **Business Rule New record** section, fill in the following fields:
    - Name: eBond Relationship Management
    - Table: eBond Relationship [u_ebond_relationship]
    - Advanced: True
3. In the **Advanced** tab, fill in the following fields:
    - When: after
    - Update: True
    - Delete: True
4. In the **Advanced** tab, fill in the following fields:
    - Condition: new eBondIncident_AlphaCo().checkCondition();
    - Script:

```
(function executeRule(current, previous /*null when async*/ ) {
    var eLog = new eBondLog();
    eLog.u_direction = 'outbound';
    eLog.u_location = 'Business Rules';
    eLog.u_name = 'eBond Relationship Management';
    eLog.u_source = 'executeRule';
    eLog.u_supplier = 'Unknown';
    eLog.u_correlate_id = current.sys_id;
    eLog.u_correlate_class_name = current.sys_class_name;
    eLog.write('Info', 'Entering.');

    var dataSet = {};

    if (current.operation() == 'update' && current.u_status.changes()) {
        dataSet.status = current.getValue('u_status');
    } else if (current.operation() == 'delete') {
        dataSet.status = 'debonded';
    } else {
        eLog.write('Info', 'Exiting.');
        return;
    }

    var dataBundle = JSON.stringify(dataSet);

    var company = new GlideRecord('core_company');
    company.addQuery('sys_id', current.getValue('u_company'));
    company.query();

    if (company.next()) {
```

```
        var supplier = company.getValue('stock_symbol').toString();
        try {
            gs.eventQueue('ebond.relationship.management.' + supplier, current,
            eLog.write('Info', 'Queued event ebond.relationship.management.' + s
        } catch (ex) {
            var message = ex.message;
            eLog.write('Medium', 'Caught exception: ' + message);
        }
    }

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

})(current, previous);
```

5. Click **Submit**.

Event Registry

Component Type: Custom

Each supplier will need it's own registered event within ServiceNow for relationship changes. This is an asynchronous event that will be handled by a **Script Action** that is defined below.

You will need to perform these instructions for AlphaCo, BetaCo, and GammaCo; replacing the supplier stock symbol as you go as the **Business Rule** defined above uses the supplier's stock symbol to generate a portion of the event name.

Instructions:

1. Navigate to **System Policy** > **Events** > **Registry**, click **New**.
2. In the **Event Registration New record** section, fill in the following fields:
    - Event Name: ebond.relationship.management.
3. Click **Submit**.

Script Actions

Component Type: Custom

Script Actions monitor for registered events being sent. When an event is created and script action associated with the event is activated.

You will need to perform these instructions for AlphaCo, BetaCo, and GammaCo; replacing the supplier stock symbol as you go.

Instructions:

1. Navigate to **System Policy** > **Events** > **Script Actions**, click **New**.
2. In the **Event Registration New record** section, fill in the following fields:
    - Name: eBond Relationship Management SUPPLIER
    - Event name: ebond.relationship.management.
    - Script:

```
// build out the data payload for SUPPLIER
// store it in u_ebond_rest_payloads to be sent to the SUPPLIER
```

```
var operation = event.parm1;
var dataBundle = event.parm2;
var returnStatus = new eBondIncident_SUPPLIER().preparePayload(operation, dataBu
```

3. Click **Submit**.

Outro

You have now successfully setup a multi-source eBonding framework. Congratulations.