Supplier Onboard

About

These are the ServiceNow instructions to onboard a new supplier for multi-source eBonding.

> **Foward Notice**
>
> There are several places where **{Supplier}** is used as a placeholder for the name of supplier. This should be replaced by the the **Stock symbol** abbreviated short form of the supplier name as defined in the **Company** record.

eBond Account

Create an account that the supplier will use to eBond with your instance. The supplier will use this account to invoke RESTful calls needed to establish an eBond.

1. Navigate to **User Administration** > **Users** from the filter navigator.
2. Either open an existing user record or create a new user record by clicking **New**. For new records, you will have to provide a user name and password. Make sure the password adheres to your company's policies. It is good practice to use a common label for the **User ID** to denote user records that are used for eBonding; e.g., ACME.eBond. Also the **First name** of the user record should reflect the supplier's name, while the **Last name** should be **eBond**. This is just a helpful suggestion and not mandatory to do so.
3. Select **Web service access only**.
4. Save the record by right-clicking in the record header and selecting **Save**.

5. Under the **Roles** tab, click **Edit**.
6. In the **Collection** filter, enter **eBond Account**.
7. Add the **eBond Account** role in the selection list to the **Roles List** by either doubled licking the role or selecting the role and clicking **>**.
8. Click **Save**.
9. Click **Update**.

> If at any point you need to terminate eBonding with a supplier you can either:
>
> - Deactivate the user record
> - Remove the **eBond Account** role It is not suggested to delete the user record. Doing so will result in unknown impacts to records that reference the user record.

Company Record

Associate the eBond account just created to the company record. This will keep record of what account is used by the supplier for eBonding in the company record.

1. Navigate to **Organization** > **Companies** from the filter navigator.
2. Either open an existing company record or create a new company record by clicking **New**.
3. Select the **eBonded** checkbox.
4. Search in the **eBond Account** field for the account that was defined earlier.
5. Fill in the short form of the **Stock symbol** field for an abbreviated supplier name.
6. Click **Update** or **Submit**.

> If at any point you need to terminate eBonding with a supplier you can either:
>
> - Deactivate the company record
> - Unselect the **eBonded** checkbox It is not suggested to delete the company record. Doing so will result in unknown impacts to records that reference the company record.

Assignment Group(s)

One of the ways a ticket is automatically eBonded with a supplier is when a ticket is assigned to an assignment group associated with the supplier. Here you will create one or more assignment groups linked to the supplier.

1. Navigate to **User Administration** > **Groups** from the filter navigator.
2. Either open an existing group record or create a new group record by clicking **New**.
3. Make sure that the supplier's company record is selected for the **Company** field.
4. Click **Save** or **Update**.

> If at any time a new supplier is chosen to provide IT services for an existing assignment group, you can change the **Company** field for the assignment group. Net new changes to the records will be eBonded automatically to the new supplier. In future releases, supplier change over will need to be accounted for for automatic true-up.

Default Service (optional)

It is the opinion of this author that all tickets in ServiceNow reference an appropriate service. If a service associated with the supplier does not already exist, create a default service that can be used for the supplier in the event the supplier does not supply a service field value. This will be used later when registering the supplier.

1. Navigate to **Configuration** > **Services** and click **New**.
2. In the **Service New record** section, fill in the following fields:
    - Name: The name of the service.
    - *: The rest of the fields should be filled out per company policy on defining services.
3. Click **Submit**.

Default Service Offering (optional)

It is the opinion of this author that all tickets in ServiceNow reference an appropriate service offering. The service offering must be a child of the aforementioned service. If a service offering associated with the supplier does not already exist, create a default service offering that can be used for the supplier in the event the supplier does not supply a service offering field value. This will be used later when registering the supplier.

1. Navigate to **Configuration** > **Services** and open the service record that will contain the service offering.
2. From the **Offerings** tab, click **New**.
3. In the **Offering record** section, fill in the following fields:
    - Name: The name of the service offering.
    - *: The rest of the fields should be filled out per company policy on defining services.
4. Click **Submit**.

eBond Registry

Create the following records in the **eBond Registry**. These values are required for various eBond operations.

Supplier	Key	Value	Note
	default.assignment.group		Reference to assignment group sys_id for the supplier.

Supplier	Key	Value	Note
	default.service		Reference to the default service for the supplier.
	default.service.offering		Reference to the default service offeringfor the supplier.
	inbound.account		Links the account to the supplier
	incident.url.endpoint		Supplier eBond link for incidents.
	incident.url.link		Supplier link to ticket. Should contain a CORRELATION_ID or CORRELATION_NUMBER tag in the name for direct linking to supplier tickets.

Event Registry

Each supplier will need it's own registered events within ServiceNow. These are asynchronous events that will be handled by a **Script Action** for that supplier.

Incident Registry

The first event handles changes made to the incident table and triggered by the business rule **eBond Incident Outbound**.

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**.

Relationship Registry

The second event handles changes made to the eBond relationship table table and triggered by the business rule **eBond Relationship Management**.

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

Each of the event registries will need a script action to handle the event. The purpose of the script action is to capture the event operation and the data payload, then invoke a supplier script include that will build out the supplier's specific data payload.

Incident Event Script Action

1. Navigate to **System Policy** > **Events** > **Script Actions**, click **New**.
2. In the **Script Action 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, dataBundle);
```

1. Click **Submit**.

Relationship Event Script Action

1. Navigate to **System Policy** > **Events** > **Script Actions**, click **New**.
2. In the **Script Action New record** section, fill in the following fields:
   - Name: eBond Relationship Management
   - 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, dataBundle);
```

1. Click **Submit**.

Script Inlcude

The script actions 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.

The example below can act as a good template to build from and not meant as must. There are several functions that are cross referenced by other external invocations, so make sure any function changes are reflected back to the callers. You will need to update the script accordingly based on the findings provided by the supplier on how to bundle data based on supplier specifications.

> The script contains **{SUPPLIER}** tags in all uppercase. This should be replaced with the supplier's name.

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_{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.getValue('sys_id');
        this.eLog.u_correlate_class_name = current.getValue('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.getValue('sys_id');
        } else {
            this.eLog.write('High', 'Cannot find the supplier\'s (' + supplier + ') comp
        }

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

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

    // func: checkCondition
    // desc: determines if the response is from {SUPPLIER}; called by ecc queue business
    // 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.getValue('u_value') == current.getValue('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 group or
    // 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 still associ
        var arrUtil = new ArrayUtil();
        var arr = current.getValue('u_ebonded_with').split(',');
        var pos = arrUtil.indexOf(arr, this.company);
        if (pos < 0 && current.getElement('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.getValue('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.getValue('number');
        data.ExternalRef = current.getValue('sys_id');

        for (var key in incidentData) {
            var map = dataMap.getSupplierValue(this.supplier, 'outbound', 'incident', ke
            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('name');
                    }
                    break;
                case "category":
                    data.Category = map.value;
                    break;
                case "subcategory":
                    data.Subcategory = map.value;
                    break;
                default:
                    break;
            }
        }

        data.CommentHistory = current.comments.getJournalEntry(-1); // full comment hist
        data.WorkNoteHistory = current.work_notes.getJournalEntry(-1); // full work note

        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();

        // IMPORTANT THIS MUST BE CALLED
        // load REST payload into the pool to be processed
        var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relationship'

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

    // func: createOperation
    // desc: ServiceNow has updated an existing record, but this is a new record for the
    // 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.getValue('number');
        data.ExternalRef = current.getValue('sys_id');

        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', 'incident', 'st
        data.Impact = dataMap.getSupplierValue(this.supplier, 'outbound', 'incident', 'i
        data.Urgency = dataMap.getSupplierValue(this.supplier, 'outbound', 'incident',
        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', 'incident',
        data.CloseNotes = current.getValue('close_notes');
        data.CloseCode = dataMap.getSupplierValue(this.supplier, 'outbound', 'incident'
        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 history
        data.WorkNote = current.work_notes.getJournalEntry(-1); // full work note histor
        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.getValue('sys_id');
        relationship.u_status = 'eBonded';
        relationship.u_state = current.getValue('state');
        relationship.u_out = 'wait';
        relationship.u_company = this.company;
        relationship.insert();

        // IMPORTANT THIS MUST BE CALLED
        // load REST payload into the pool to be processed
        var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relationship

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

    // func: updateOperation
    // desc: ServiceNow has updated the record, update the {SUPPLIER} records as appropr
    // 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.getValue('sys_id'));
        relationship.addQuery('u_status', 'ebonded');
        relationship.addQuery('u_company', this.company);
        relationship.query();
        while (relationship.next()) {

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

            // if reflect, then perform data mapping
            if (relationship.u_reflect == true) {
                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;
                    }
                }
            } 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.getDisplayValue
                        }
                        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.getDisplayValue(
                        }
                        break;
                    case 'ci':
                        var ci = new GlideRecord('cmdb_ci');
                        ci.addQuery('sys_id', incidentData[key]);
                        ci.query();
                        if (ci.next()) {
                            comment += newline + key + ': ' + ci.getDisplayValue('na
                        }
                        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.getDisplayValu
                        }
                        break;
                    case 'service_offering':
                        var service_offering = new GlideRecord('cmdb_ci_service');
                        service_offering.addQuery('sys_id', incidentData[key]);
                        service_offering.query();
                        if (service_offering.next()) {
                            comment += newline + key + ': ' + service_offering.getDi
                        }
                        break;
                    default:
                        var map = dataMap.getSupplierValue(this.supplier, 'outbound'
                        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();
            }

            // IMPORTANT THIS MUST BE CALLED
            // load REST payload into the pool to be processed
            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relations
        }

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

    // func: deleteOperation
    // desc: ServiceNow has deleted the record, debond associated {SUPPLIER} records
    //       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.getValue('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.getValue('u_correlation_number');
            data.state = '0';
            var payload = JSON.stringify(data);

            // IMPORTANT THIS MUST BE CALLED
            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relations

            // change the relationship
            relationship.setValue('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 record
                this.updateOperation(dataBundle); // decision is to update record(s)
                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 relationship
    // 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.getValue('u_correlation_number');
            data.state = '0'; // {SUPPLIER} deBond state code
            var payload = JSON.stringify(data);

            // IMPORTANT THIS MUST BE CALLED
            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relations
        } else {
            var data = {};
            data.number = current.getValue('u_correlation_number');
            data.state = '-1'; // {SUPPLIER} re-eBond state code
            var payload = JSON.stringify(data);

            // IMPORTANT THIS MUST BE CALLED
            var eBondRest = new eBondRestPayload().load(this.company, 'u_ebond_relations
        }

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

    type: 'eBondIncident_{SUPPLIER}'
};
```

3. Click **Submit**.

REST Messages

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.

> In the instructions below there is a field called **Endpoint** with the value of **${endpoint}**, this is the true value of the field and should not be confused with {SUPPLIER} that has been replaced throughout these steps.

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:
    - 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

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.getValue('sys_id');
    eLog.u_correlate_class_name = current.getValue('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', 'incident.url.endpoint');
```

```
    url.query();
    if (url.next()) {
        if (current.getValue('source') != url.getValue('u_value')) {
            eLog.write('High', 'Security Alert. Inbound source REST message for ' + supp
            return;
        }
    } else {
        eLog.write('High', supplier + ' is missing eBond registry key incident.url.endpo
        return;
    }

    eLog.write('Debug', 'ECC Queue sys_id: ' + current.getValue('sys_id'));
    eLog.write('Debug', 'ECC Queue topic: ' + current.getValue('topic'));
    eLog.write('Debug', 'ECC Queue state: ' + current.getValue('state'));
    eLog.write('Debug', 'ECC Queue payload: ' + current.getValue('payload'));
    eLog.write('Debug', 'ECC Queue response to: ' + current.getValue('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-codes.xhtm
    // 1xx: Informational - Request received, continuing process
    // 2xx: Success - The action was successfully received, understood, and accepted
    // 3xx: Redirection - Further action must be taken in order to complete the request
    // 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
    // 5xx: Server Error - The server failed to fulfill an apparently valid request

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

    // process the next REST message for the ticket
```

```
    if (executeNext) {
        var next = new eBondRestPayload().executeNext(relationship.getValue('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.getValue('u_source'));

    // update the incident u_ebonded_with
    // traverse all related eBonds
    var relationshipChk = new GlideRecord('u_ebond_relationship');
    relationshipChk.addQuery('u_source', relationship.getValue('u_source'));
    relationshipChk.addQuery('u_status', 'ebonded');
    relationshipChk.addQuery('u_company', company.getValue('sys_id'));
    relationshipChk.query();

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

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

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

5. Click **Submit**.