**Event Subscriber**

Event types in Business Central involve understanding how to use **events** to extend and customize the application without modifying the base application code. Events are categorized into different types, and here’s a guide to help you practice with them:

**1. Publisher and Subscriber Model**

* **Publisher Events**: These are events that are triggered in the base application or by your own custom code. They include:
  + **Integration Events**: Used for integration purposes. They do not impact the execution of the code.
  + **Business Events**: Used to trigger business logic. They allow you to react to certain business conditions.
  + **Trigger Events**: Specific to table or page triggers (e.g., OnInsert, OnModify).
* **Subscriber Events**: These are your custom procedures that respond to published events. You subscribe to these events to execute your code when the event is triggered.

**2. Common Event Types**

* **OnBefore Events**: These events are triggered before the main logic of a procedure is executed. You can use these to modify behavior or cancel the operation.
  + Example: OnBeforeInsertEvent, OnBeforeValidateEvent
* **OnAfter Events**: These events are triggered after the main logic of a procedure is executed. These are used for post-processing logic.
  + Example: OnAfterInsertEvent, OnAfterModifyEvent
* **Integration Events**: These can be called from any part of the application and are typically used to extend logic without affecting the core functionality.
  + Example: OnAfterPostingSalesInvoiceEvent
* **Business Events**: These are used to encapsulate business logic that other modules might need to respond to.
  + Example: OnCustomerBlockedEvent

**3. How to Practice Events**

* **Step 1: Create Event Publisher**
  + If you want to create custom events, define them in your codeunit with the [EventPublisher] attribute.
  + Example:

|  |
| --- |
| [IntegrationEvent(false, false)]  local procedure OnBeforeCalculateTax(var SalesHeader: Record "Sales Header");  begin  end; |

* **Step 2: Create Event Subscriber**
  + Use the [EventSubscriber] attribute to subscribe to events.
  + Example:

|  |
| --- |
| [EventSubscriber(ObjectType::Table, Database::"Sales Line", 'OnBeforeInsertEvent', '', false, false)]  local procedure MyOnBeforeInsertEvent(var Rec: Record "Sales Line"; var xRec: Record "Sales Line"; RunTrigger: Boolean);  begin  // Custom logic here  end; |

* **Step 3: Practice Different Events**
  + **OnBefore/OnAfter Events**: Try subscribing to OnBeforeInsertEvent or OnAfterModifyEvent of a table.
  + **Integration Events**: Create an integration event in one codeunit and subscribe to it in another.
  + **Business Events**: Use business events to manage complex logic, such as handling the blocking of a customer across modules.

**4. Best Practices**

* **Avoid Direct Modifications**: Always use events to extend the base application instead of directly modifying base objects.
* **Test Extensively**: Ensure that subscribing to an event does not negatively impact performance or cause unintended behavior.
* **Use IsHandled Pattern**: For OnBefore events, use the IsHandled pattern to optionally skip the default behavior.
  + Example:

|  |
| --- |
| [EventSubscriber(ObjectType::Table, Database::"Sales Line", 'OnBeforeInsertEvent', '', false, false)]  local procedure MyOnBeforeInsertEvent(var Rec: Record "Sales Line"; var IsHandled: Boolean);  begin  IsHandled := true; // Skip the standard insert behavior  end; |

**5. Sample Exercise**

* **Exercise 1**: Create a subscriber for the OnAfterPostSalesInvoice event and log some custom information to a custom table.
* **Exercise 2**: Create a custom event in a codeunit and trigger it from a page action, then create a subscriber in another codeunit.

By practicing these steps, you’ll gain a solid understanding of how to work with events in Business Central, allowing you to effectively extend and customize the application. Let me know if you want to work on a specific type of event or need further guidance on any of the steps!

**Syntax:**

|  |
| --- |
| [EventSubscriber(ObjectType, ObjectName, EventName, ElementName, SkipOnMissingLicense, SkipOnMissingPermission)] |

**1. ObjectType (ObjectType::)**

This defines the **type** of object where the event is declared. It tells the subscriber which type of object contains the event. Some common object types are:

* Codeunit: If the event is declared in a **Codeunit**.
* Table: If the event is declared in a **Table**.
* Page: If the event is declared in a **Page**.
* Report: If the event is declared in a **Report**.

**Example:**

|  |
| --- |
| ObjectType::Codeunit  ObjectType::Table  ObjectType::Page |

This tells the compiler that the event is in a codeunit, table, or page.

**2. ObjectName (ObjectType::"ObjectName")**

This defines the **specific object** where the event resides. It’s important to reference the correct object that contains the event you want to subscribe to. It can be a system object or a custom one.

For example:

* Codeunit::"Sales-Post": This refers to the standard Sales-Post codeunit.
* Table::"Customer": This refers to the **Customer** table.
* Page::"Sales Order": This refers to the **Sales Order** page.

**Example:**

|  |
| --- |
| Codeunit::"Sales-Post"  Table::"Customer"  Page::"Sales Order" |

This tells the subscriber where to look for the event.

**3. EventName ('EventName')**

This is the **name of the event** you are subscribing to. Business Central objects often have multiple events that you can subscribe to. You must specify which one you want to react to.

Events are typically named descriptively. Examples include:

* OnAfterInsert: Triggered after a record is inserted.
* OnBeforePostSalesHeader: Triggered before posting sales headers.
* OnAfterPostSalesLines: Triggered after posting sales lines.

**Example:**

|  |
| --- |
| 'OnAfterPostSalesLines'  'OnAfterInsert'  'OnBeforePostSalesHeader' |

This is the specific event you're subscribing to.

**4. ElementName ('ElementName')**

This parameter is used if you want to subscribe to an event related to a **specific field or control** within an object, typically used in **pages** or **tables**.

* In a table, you can subscribe to field-level events (like OnValidate or OnLookup).
* In a page, you can subscribe to control-level events (like OnOpenPage).

If you're not subscribing to a field-specific or element-specific event, you leave this empty ('').

**Example:**

* If you're subscribing to an event related to a specific field:

|  |
| --- |
| 'Customer No.' |

* If it's not specific to an element:

|  |
| --- |
| '' |

**5. SkipOnMissingLicense (true/false)**

This Boolean parameter tells Business Central **whether to skip the event** if the necessary license is not available.

* true: The event subscriber will be **skipped** if the license is missing.
* false: The event subscriber will **not** be skipped, even if the license is missing.

**Example:**

Typically, you set this to false unless you want to conditionally execute based on the license.

|  |
| --- |
| true  false |

**6. SkipOnMissingPermission (true/false)**

This Boolean parameter tells Business Central **whether to skip the event** if the user does not have the required permissions.

* true: The event subscriber will be **skipped** if the user lacks the required permissions.
* false: The event subscriber will **not** be skipped, even if the user lacks the required permissions.

**Example:**

|  |
| --- |
| true  false |

You would usually set this to false if the event should always be executed regardless of permissions, but if there’s sensitive data, you might want to set it to true.

**Full Example Explained:**

|  |
| --- |
| [EventSubscriber(ObjectType::Codeunit, Codeunit::"Sales-Post", 'OnAfterPostSalesLines', '', false, false)] |

* **ObjectType**: Codeunit — The event is in a codeunit.
* **ObjectName**: "Sales-Post" — The specific object is the Sales-Post codeunit.
* **EventName**: 'OnAfterPostSalesLines' — The subscriber listens for the OnAfterPostSalesLines event.
* **ElementName**: '' — No specific element or field within the codeunit is being targeted, so this is left empty.
* **SkipOnMissingLicense**: false — The event should still execute even if the license is missing.
* **SkipOnMissingPermission**: false — The event should still execute even if the user lacks specific permissions.

**Example: Subscribing to a Table Event**

|  |
| --- |
| [EventSubscriber(ObjectType::Table, Table::"Sales Line", 'OnAfterInsertEvent', '', false, false)] |

* **ObjectType**: Table — The event is in a table.
* **ObjectName**: "Sales Line" — The specific object is the Sales Line table.
* **EventName**: 'OnAfterInsertEvent' — The subscriber listens for the OnAfterInsertEvent.
* **ElementName**: '' — No specific field is targeted.
* **SkipOnMissingLicense**: false — The event will not be skipped if the license is missing.
* **SkipOnMissingPermission**: false — The event will not be skipped if the user lacks permission.

**Example: Subscribing to a Field Event in a Table**

|  |
| --- |
| [EventSubscriber(ObjectType::Table, Table::"Customer", 'OnValidate', 'Name', false, false)] |

* **ObjectType**: Table — The event is on a table.
* **ObjectName**: "Customer" — The specific object is the Customer table.
* **EventName**: 'OnValidate' — The subscriber listens for the OnValidate event.
* **ElementName**: 'Name' — The event is specific to the Name field.
* **SkipOnMissingLicense**: false — The event should still execute if the license is missing.
* **SkipOnMissingPermission**: false — The event should still be executed if the user lacks permissions.

**Example**

Create a code unit in a project and add the following code. This code basically is called when we post a document in Sales Order. 'OnAfterPostSalesLines' is the event name from the code unit ‘Sales-Post’ which will run the following code after the action ‘Post’ in clicked.

|  |
| --- |
| codeunit 50100 "Sales Order Release Notificat"  {      [EventSubscriber(ObjectType::Codeunit, Codeunit::"Sales-Post", 'OnAfterPostSalesLines', '', false, false)]      local procedure OnAfterPostSalesLines(SalesHeader: Record "Sales Header")      var          Notification: Notification;      begin          // Create and send a notification          Notification.Message := 'Sales lines have been successfully posted for Order No. ' + SalesHeader."No.";          Notification.Send();      end;  } |

|  |
| --- |
|  |

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated |  |

|  |
| --- |
|  |