

Microsoft Dynamics® AX 2012

Eventing

White Paper

This document describes the concept of events and how they can be used in Microsoft Dynamics AX.

Date: January 2011

<http://microsoft.com/dynamics/ax>

Author: Peter Villadsen, Senior Program Manager for X++ language

Send suggestions and comments about this document to adocs@microsoft.com. Please include the title with your feedback.



Table of Contents

Overview	3
The nature of events	3
Terminology	4
Coded events	4
Automatic events	5
Method call before and after events	5
Event handlers	6
Adding handlers in the AOT	6
Adding handlers in code.....	6
Tips for events	7
Do	7
Don't.....	7

Overview

Microsoft Dynamics® AX is rarely used by customers "out of the box." Each customer has a set of requirements to implement in the product. This has led to creation of an ecosystem of partners that customize the product to fit individual customers. The complexity of the customization process translates into cost for the customer. Currently, partners invest significant time in browsing and understanding the source code that is shipped by Microsoft. When the need for a customization is identified, the partner typically modifies the code in the customer's layer. The runtime engine ensures that the customization takes effect.

A problem with the existing customization model is that customizations can be broken when Microsoft restructures code in a lower layer of the product, when a newer version is released. Fixing earlier customizations of the next version upgrade can be expensive for partners.

The use of events has the potential to lower the cost of creating and upgrading customizations. Developers who create code that is often customized by others should create events in places where customizations typically occur. Then developers who customize the original functionality in another layer can subscribe to an event. When the customized functionality is tied to an event, then the underlying application code can then be rewritten and have little impact on the customization, as long as the same events are raised in the same sequence from one version to the next.

The nature of events

Object oriented programming paradigms often map easily to the problem domains that they attempt to model. For example, classes are entities that typically represent a type of thing in real life, and the names chosen for classes are typically nouns like Accounts, People, and Customers. Methods are used to make changes to objects, and the names chosen for methods are then typically verbs that specify what operation is done on the objects, like Deposit, Hire, and Invoice.

An event can be said to represent something that happens within an object. Applicable event names are verbs in the past tense, like Deposited, Hired, and Invoiced.

Events are a simple and powerful concept. We encounter events all the time in our daily lives: The phone rings, an email message arrives, a computer breaks down because coffee is spilled on it, or you win the lottery. Some events are expected, but some are not. Many events happen all around us all the time and some are outside of the scope of our interest while some require us to take some sort of action.

Events can be used to support the following programming paradigms:

- **Observation:** Events can be used to look for exceptional behavior and generate alerts when such behavior occurs. An example of this type of event is regulation compliance systems. For example, if more than a designated amount of money is transferred from one account to another, the system triggers a workflow that solicits acceptance of the transaction.
- **Information dissemination:** Events can deliver the right information to the right consumers at the right time. Information dissemination is supported by publishing an event to anyone wishing to react to it.
- **Decoupling:** Events produced by one part of the application can be consumed by a completely different part of the application. There is no need for the producer to be aware of the consumers, nor do the consumers need to know details about the producer. One producer's event can be acted upon by any number of consumers. Conversely, consumers can act upon any number of events from many different producers.

Terminology

We modeled the concept of events in Microsoft Dynamics AX 2012 on .NET eventing concepts.

The following table lists the terms in Microsoft Dynamics AX 2012 related to events.

Term	Definition
Producer	The <i>producer</i> is the logic that contains the code that causes a change. It is an entity that emits events.
Consumer	The <i>consumer</i> is the application code that represents an interest in being notified when a specific event occurs. It is an entity that receives events.
Event	An <i>event</i> is a representation of a change having happened in the producer.
Event Payload	The <i>event payload</i> is the information that the event carries with it. When a person is hired, for example, the payload might include the employee's name and date of birth.
Delegate	A <i>delegate</i> is the definition of the information passed from the producer to the consumer when an event takes place.

Coded events

In X++, you can add delegates as members of a class. The syntax for defining a delegate is the same as the syntax used for defining a method, except that:

- The "delegate" keyword is used.
- Neither "public" nor any other accessibility specifier is allowed.
- The return type must be "void."
- The body must be empty; that is, it can contain neither declarations nor statements.

For example, a delegate for an event that is raised when a person is hired could be expressed like this:

```
delegate void hired(str name, UtcDateTime startingDate) {}
```

Figure 1 illustrates a class called **EventTest** that includes one delegate called **MyDelegate** that accepts a string and an integer.

Note The lightning icon indicates that this is a delegate and not an X++ method.

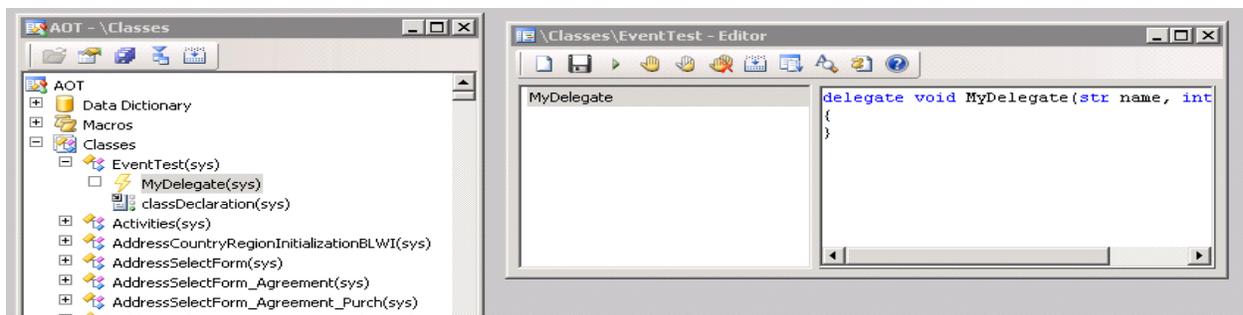


Figure 1: Application Object Tree (AOT) showing a class with an event

When the application code needs to signal that a new person has been hired, it uses the delegate as a method call, supplying the event payload as parameters, as shown below:

```
{
  while select * from Employees where hiredDate == today
  {
    this.hired(employees.Name, employees.FirstWorkingDay);
  }
}
```

The parameters defined in the parameter profile can be any type allowed in X++. In particular, it is useful to pass an object instance, and to have the handlers modify the state of the object. In this way, the publisher can solicit values from the subscribers.

Event handlers are run within the caller's transaction context.

Subscribers can be added to the event in one of two ways: in the AOT or in X++ code.

Automatic events

Automatic events are events that are triggered by the environment as the result of something happening. Currently, a developer can indicate that an event is triggered when a method is called, as described below.

Method call before and after events

Pre and Post are predefined events that occur when methods are called. The *pre-event handlers* are called before the execution of the designated method, and the *post-event handlers* are called after the method call has ended. It may be a useful mental model to understand this as augmenting the existing method with methods called before and after the normal body, as shown in Table 1.

Not having any event handlers for a particular method leaves the method intact. Therefore, there is no overhead incurred for methods that do not have any pre or post handlers assigned to them.

The example in Table 1 also demonstrates that if an exception is thrown in any of the pre-event handlers, neither the remaining handlers nor the method itself is invoked. If a method that has pre-event and/or post-event handlers throws an exception, the remaining post-event handlers are not invoked. If an exception is thrown in a post-event handler, the remaining handlers are not called. In any case, the exception propagates back to the caller as normal.

Method before adding handlers	Method after adding handlers
<pre>void MyMethod (int i) { ... Method Body ... }</pre>	<pre>void MyMethod (int i) { PreHandler1(i); PreHandler2(i); ... Method Body ... PostHandler1(i); PostHandler2(i); }</pre>

Table 1: Methods with and without handlers

Each of the pre-event handlers can access the original values of the parameters and modify them as required. The post-event handlers can modify the return value of the called method.

Event handlers

Event handlers are the methods that are called when the delegate is called, either directly through code (for the coded events) or from the environment (in the modeled events). The relationship between the delegate and the handlers can be maintained in the code or in the Application Object Tree (AOT).

Adding handlers in the AOT

A developer must identify a static method to handle the event on the delegate. Only static methods can be used in this context. Event handlers can be added to the delegate by simply dragging the event handler to the delegate node that represents the event to be handled. The handler can be removed using the **Delete** menu item available for any node in the AOT.

Figure 2 shows an event handler that has subscribed to an event:

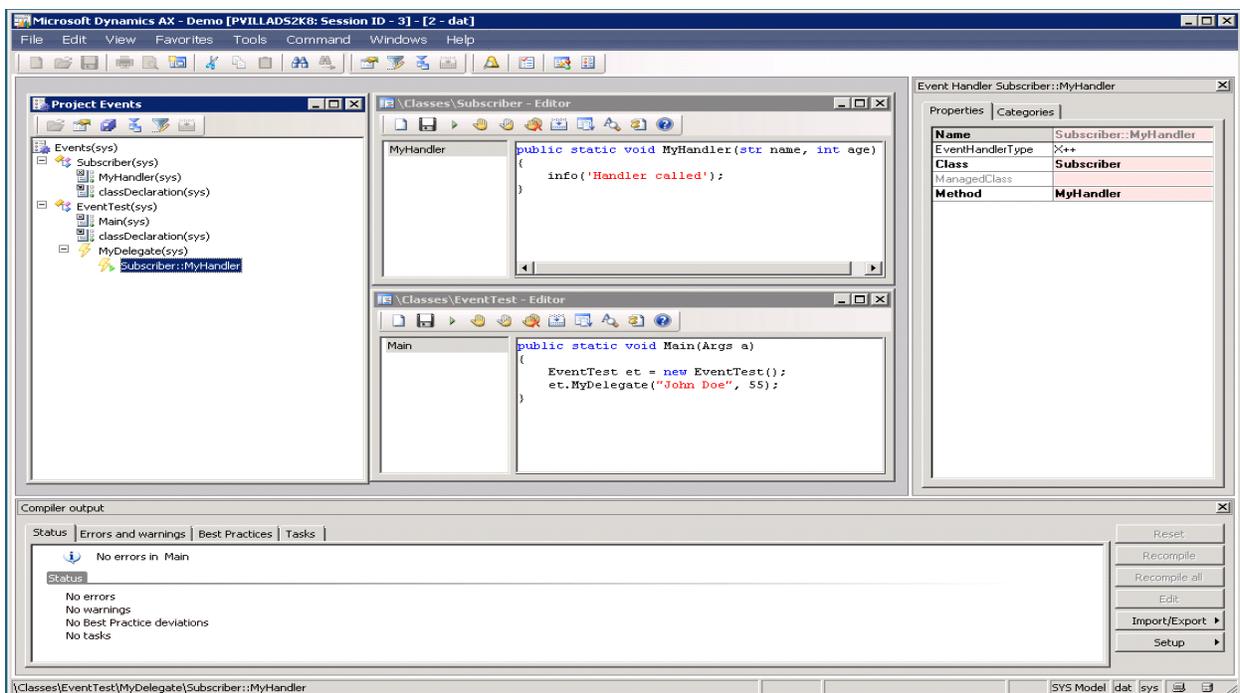


Figure 2: X++ Event handler added to MyDelegate

The property sheet for an individual event handler shows the class and the method to call when the event occurs. The handler can be implemented either in managed code or in X++.

Adding handlers in code

It is possible to add and remove event handlers to events by using special X++ syntax, as shown below. The delegate name appears on the left side of the += operator. On the right hand side the keyword event handler is given, along with the qualified name of the handler to add. The compiler will check that the parameter profiles of the delegate and the handler match.

```
private void AddStaticHandler()
{
    this.MyDelegate += eventhandler (Subscriber::MyHandler);
}
```

The qualified name above uses the `::` syntax to separate the type name and the delegate to designate that the handler is static. If a method on a particular object instance should be called, the dot character may be used instead of `::`, as illustrated in the following snippet:

```
private void AddInstanceHandler()
{
    Subscriber s = new Subscriber();
    this.MyDelegate += eventhandler (s.InstanceHandler);
}
```

Removing the handler from a delegate is accomplished by using the `-=` operator instead of the `+=` operator.

Tips for events

The following list summarizes DOs and DON'Ts to consider when working with events.

Do

- Define events for things that programmers who are customizing the behavior of the system should react to. You should identify natural customization points and use events to publish the relevant information to subscribers.
- Maintain the semantics implied by the event from one version to the next. Part of the usefulness of events comes from the preservation of the semantics of the events. Once an interface has been defined, third parties will be able to leave their existing code unmodified if the semantics are left unchanged (even though the implementation behind might be modified).
- Document the events that you create well, for the sake of the people doing customization later.
- Favor defining events over using pre and post events on methods. Events are strong interfaces, while methods are an implementation artifact: A method's responsibilities and parameters may change over time, and they may disappear altogether as the developer sees fit.

Don't

- Do not try to raise an event from outside the class in which the event is defined. The X++ compiler disallows raising events from outside the class by design.
- Do not make customizations that rely on the order in which event handlers are executed. No guarantees are made by the runtime environment about the order in which the event handlers are called.

Microsoft Dynamics is a line of integrated, adaptable business management solutions that enables you and your people to make business decisions with greater confidence. Microsoft Dynamics works like and with familiar Microsoft software, automating and streamlining financial, customer relationship and supply chain processes in a way that helps you drive business success.

U.S. and Canada Toll Free 1-888-477-7989

Worldwide +1-701-281-6500

www.microsoft.com/dynamics

This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Microsoft Dynamics, and the Microsoft Dynamics logo are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.