# AUTOEDGE|THEFACTORY OEBPM USE-CASES

Overview of how ABL / AutoEdge|TheFactory and OEBPM / Savvion workflows interact.

## 1. WORKFLOW DEFINED AND RUN BY SAVVION

Savvion defines the entire vehicle build process – the order of the individual steps - and calls an ABL Appserver/WebService per workstep. This allows visibility, responsiveness, etc into the workflow/process. It requires the ABL application workflow to be sufficiently decomposed / modularized so that each step can be called in isolation.

As a variation, the individual ABL steps can be re-entrant into the Savvion process via the BizLogic API. Typically this would entail updating some dataslot values.

The AETFMapping class contains the mappings/bindings from service names to concrete classes. The mappings/bindings below are used in all the use-cases, although not all use-cases use all mappings/bindings. The ServiceManager resolves the mapping from a service name to the actual class used to perform that service.

A ServiceProvider is an object that takes a service message (data fetch or save request, or workflow request) and satisfies it.

```
/* Bindings resolved by InjectABL Kernel */
AETFMapping.cls
  Bind("IServiceProvider"):Named("BuildVehicleWorkFlow")
    :To("AutoEdge.WorkFlow.BuildVehicle").

  /* 2 bindings for different entry points. The work done is same */
  Bind("IServiceProvider"):Named("WorkStep-BuildVehicle-ProcessComponent")
    :To("AutoEdge.WorkStep.ProcessComponent").
  Bind("IProcessComponent")
    :To("AutoEdge.WorkStep.ProcessComponent").

  Bind("IServiceProvider"):Named("WorkStep-BuildVehicle-CompleteVehicleBuild")
    :To("AutoEdge.WorkStep.CompleteVehicleBuild").
  Bind("ICompleteVehicleBuild")
    :To("AutoEdge.WorkStep.CompleteVehicleBuild").

  Bind("INotifySavvion")
    :To("WorkFlow.SavvionNotifier").
```

The code below is an example of a single WorkStep as exposed to  Savvion via the WebService/AppServer. In this example, the workstep is fire-and-forget as far as Savvion's concerned. The ABL code has no way of knowing that the request was made by Savvion. Of course, that could be remedied by passing some Savvion-specific context (like a Process Instance Id or piid) in which case the ABL code could call back to that process instance.

```
/* .P is exposed via WebServices to Savvion */
workstep_buildvehicle_processcomponent.p
    (input order-id)
  /* uses AETFMappinge.cls for mapping to concrete classes */
  oSP = ServiceMgr:GetServiceProvider("WorkStep-BuildVehicle-ProcessComponent")

  oReq = new WorkflowRequest()
  oReq:SetMessageData(order-id)
  oResp = oSP:ExecuteRequest(oReq).
```

The “WorkStep-BuildVehicle-ProcessComponent” mapping resolves into an instance of a WorkStep.ProcessComponent class, which performs the necessary work for that workstep in the workflow. Once the “ProcessComponent” work is complete, control returns back to the controlling Savvion process, which decides what step to perform next.

```
AutoEdge.WorkStep.ProcessComponent.cls : IServiceProvider, IProcessComponent
  /* IServiceProvider entry-point */
  ExecuteRequest(WorkflowRequest)
    this-object:ProcessComponent()

  /* IProcessComponent entry-point */
  ProcessComponent()
    /* do stuff */
```

## 2. WORKFLOW DEFINED BY AN ABL APPLICATION AND RUN BY SAVVION

The ABL application defines the entire vehicle build process. This can be hardcoded as method/procedure calls, or can be stored in a repository of sorts. While the vehicle build process is a single workstep in Savvion,  it is effectively an external sub-process that Savvion will invoke via an ABL Appserver/WebService.

As a variation, the individual ABL steps can be re-entrant into the Savvion process via the BizLogic API. Typically this would entail updating some dataslot values in cases where the Savvion process waits for the ABL workflow to complete; alternatively, the ABL application will run to completion, blocking the Savvion process until it does.

```
/* .P is exposed via WebServices to Savvion */
service_buildvehicle.p
 (input order-id, input piid)

  /* uses AETFMappinge.cls for mapping to concrete classes */
  oSP = ServiceMgr:GetServiceProvider("BuildVehicleWorkFlow")

  oReq = new SavvionWorkflowRequest()
  oReq:ProcessInstanceId = piid.
  oReq:SetMessageData(order-id)
  oResp = oSP:ExecuteRequest(oReq).
```

The “BuildVehicleWorkFlow” mapping resolves into an instance of a WorkFlow.StandardBuildVehicle class, which performs the necessary work for that workstep in the workflow. Once the “ProcessComponent” work is complete, control returns back to the controlling Savvion process, which decides what step to perform next.

```
AutoEdge.WorkFlow.StandardBuildVehicle.cls : IServiceProvider, IBuildVehicle
  /* IServiceProvider entry-point */
  ExecuteRequest(WorkflowRequest)
    this-object:BuildVehicle()
    /* We may want to tell Savvion that we're done */
    if type-of(WorkflowRequest, ISavvionRequest) then
        oNotifier = ServiceMgr:StartService('INotifySavvion')
        oNotifier:NotifySavvion(WorkflowRequest:ProcessInstanceId, ...)

  /* IBuildVehicle entry-point */
  /* order of steps hardcoded or temp-table driven; doesn't really matter */
  BuildVehicle()
    o = ServiceMgr:StartService('IProcessComponent')
    o:ProcessComponent()
    ...
    o = ServiceMgr:StartService('ICompleteVehicleBuild')
    o:CompleteVehicleBuild()
    /* done building vehicle */
```