Overview of Action Plugins

The **iPaper.NET** architecture allows to create and install *action plugins.* An *action plugin* should implement the **IAction** interface. A class which implements the IAction interface should be decorated with the **ActionAttribute** attribute. The iPaper.Actions.dll contains all data types required to write an *action plugin* for the **iPaper.NET**.

The IAction Interface

```
public interface IAction : IDisposable {
    // general properties
    Guid Guid { get; }
    string Name { get; }
    string Description { get; }

    // user interface
    System.Windows.Forms.Control ReducedUI { get; }
    System.Windows.Forms.Control FullUI { get; }

    void Init();
    void Run(string filePath, ActionContext context);
    void Done();

    void SetProperty(string name, object value);
    object GetProperty(string name);

    void ReadFromXml(System.Xml.XmlElement el);
    void WriteToXml(System.Xml.XmlWriter writer, bool writeElement);
}
```

The ActionContext Class

```
public class ActionContext {
    public IList<ActionException> Errors { get; private set; }

    public ActionContext();
    public bool IsBatch { get;set; }
    public string ActionDefinition { get; set; }
    public System.Xml.XmlElement CreateActionDefinitionElement();
    public Stationery.StationeryCollection Stationeries { get; set; }
}
```

Workflow

At startup, the **iPaper.NET** scans the installation folder for assemblies which contains *action plugins*. An *action plugin* should be named iPaper.Actions.<actionname>.dll. It is recommended to create a separate assembly for each *action plugin.* All types which implement the **IAction** interface and have the **ActionAttribute** attribute will be created (one instance of each *action plugin* will be created).

When a file should be processed, the **iPaper.NET** will call the members of an *action plugin* in the following order:

1.) creates an instance of the plugin (ctor)
2.) Init(): initialization of the action should be done at this point
3.) Run(filePath, actionContext): the action implementation should be performed at this point
4.) Done(): uninitialization should be performed
5.) Dispose(): all resources owned by the action should be disposed (this method is inherited from IDisposable)

Sample code for implementing the Run method:

```
public void Run(string filePath, ActionContext context) {
  // check whether the action is called from batch
  if (context.IsBatch && !string.IsNullOrEmpty(context.ActionDefinition)) {
    // create the XML element which defines the action
    System.Xml.XmlElement actionElement = context.CreateActionDefinitionElement();
    if (actionElement == null)
      throw new ActionException("Failed to read the action definition");

    // read the action definition from the XML element
    ReadFromXml(actionElement);

    bool result = false;
    // do processing
    // result = DoProcessing(...)
    if (!result)
      throw new ActionException("Failed to execute the action");
  }
  else {
    // interactive processing
    new NotImplementedException("The interactive processing was not implemented");
  }
```