Developing workflows with GWE

Contents

[GWE Components 2](#_Toc346277652)

[Anatomy of a GWE Workflow 3](#_Toc346277653)

[GWE Behavioral Flow Chart 5](#_Toc346277654)

[The Templating Language 5](#_Toc346277655)

[GWE Configuration Screen Example 7](#_Toc346277656)

[Workflow Variables: How they are defined and when to use them 8](#_Toc346277657)

[The GWE Actions 8](#_Toc346277658)

[Importing/Exporting/Deploying/Backing Up Workflows 9](#_Toc346277659)

[GWE SOAP Web Services 9](#_Toc346277660)

[Developer Reference 10](#_Toc346277661)

[Prerequisites 10](#_Toc346277662)

[Project structure 11](#_Toc346277663)

[How to make a custom action 11](#_Toc346277664)

The Generic Workflow Engine (GWE) is a programmable SharePoint event receiver intended to serve as a tool for building event-driven workflows in a minimalistic fashion. The theory behind GWE is that most workflows can be implemented using a single list, a timer job and an intelligent event receiver. Let a list represent a workflow. Each item inside the list represents an individual workflow instance, and therefore has its own “state” with respect to the workflow. When a given item is added or updated, a programmable event receiver performs transitions on the item, allowing the item to move between states. Using this notion of states and transitions, the system can effectively model a state diagram.



The process for implementing a workflow using GWE generally follows these steps:

1. Build a flow chart or a state diagram as a mechanism to clearly define your workflow
2. Identify and list out all the states of the workflow. Identify and list out all the transitions that connect the states together. Note what actions should occur within each of these transitions.
3. Design an InfoPath form template or a list containing all the fields necessary to implement the workflow
4. Open the list settings for the workflow list/library and program the GWE event receiver based on the information from step 2.
5. Test and deploy

# GWE Components

* **SharePoint components**
  + Solution: Single WSP. Oxbow.Gwe.SharePoint.wsp
  + Features:
    - **Farm-level timer job feature** –Executes GWE timer tasks
    - **Web-level GWE feature** – Enables the user to use GWE on lists within the web
* **Major program functionality**
  + **GWE Configuration Page –** When web-level feature is enabled, this page is found in the list settings ribbon under any list. Manage all the details of any specific workflow on this page.
  + **GWE scripting language –** Program events using macros defined in a mini-language.
  + **Event receiver –** Specialized event receiver fires on ItemAdded/ItemUpdated. It reads the workflow configuration for instructions and performs actions on list items.
  + **GWE Web Services –** Helpful web service methods to aid with InfoPath development
  + **Timer Job –** On a routine basis, this job will crawl web applications for webs and lists with GWE feature enabled and activated. For each list it will read each timer task. For each timer task, it update specific items in the list triggering the event receiver to execute specific transitions.

# Anatomy of a GWE Workflow

Each list/library has a single workflow configuration. The workflow configuration is stored as XML text inside the site property bag. A workflow configuration contains the following fields:

* **Admin email** – When something goes wrong, who gets notified? Usually we set this to a distribution group.
* **Transition field** – This represents a field in the list that will cue GWE which transition to execute during an ItemAdded/ItemUpdated event.
* **Timer Tasks** – An ordered collection of timer tasks. Each timer task contains:
  + **List view name** – A timer task executes on all items in a given list view
  + **Transition name** – For each item in the transition set the transition field to the transition name value, thereby triggering the event receiver
* **Global Variables –** A collection of key/value pairs that allow you to define variables and reference them throughout the workflow transitions, allowing for reuse of frequently referenced functions
* **Transitions** – A collection of transitions under the workflow configuration. Each transition contains:
  + **Transition name** – When an item is added/updated and the event receiver triggers and the item’s transition field value matches the transition name, all the actions inside the matching transition will execute on the list item.
  + **Actions** – An ordered collection of actions to perform on the list item when the transition executes. GWE has different types of actions, but the common parts are
    - **Action type** – one of the following types:
      * **Send email**
      * **Update infopath form fields**
      * **Update list item**
      * **Log to list**
      * **Change permissions**
      * **Run custom code**
    - **Alias –** Just a name
    - **Condition –** If the condition field is not blank, it is expected to be a GWE code block that evaluates to true/false. If the code block evalues to false, the action will not execute.
    - **Halt on failure** – If the action fails for some reason, should GWE continue running the next actions, or halt?

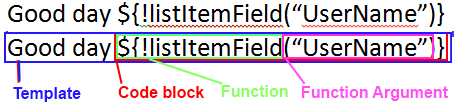


# GWE Behavioral Flow Chart



# The Templating Language

When sending emails, writing logs, or editing fields you usually need to mix data-driven values with static text to get the desired result. In GWE, the task of mixing data with text is called templating, and it is supported through the use of a small language syntax based on the [NVelocity templating language](http://nvelocity.sourceforge.net/) which mixes text and code blocks. Code blocks are wrapped in ${…} brackets and spread throughout the text. If you has a list item field called “UserName” which had a value “John” then you could create a template message “Good day John” with the following syntax:

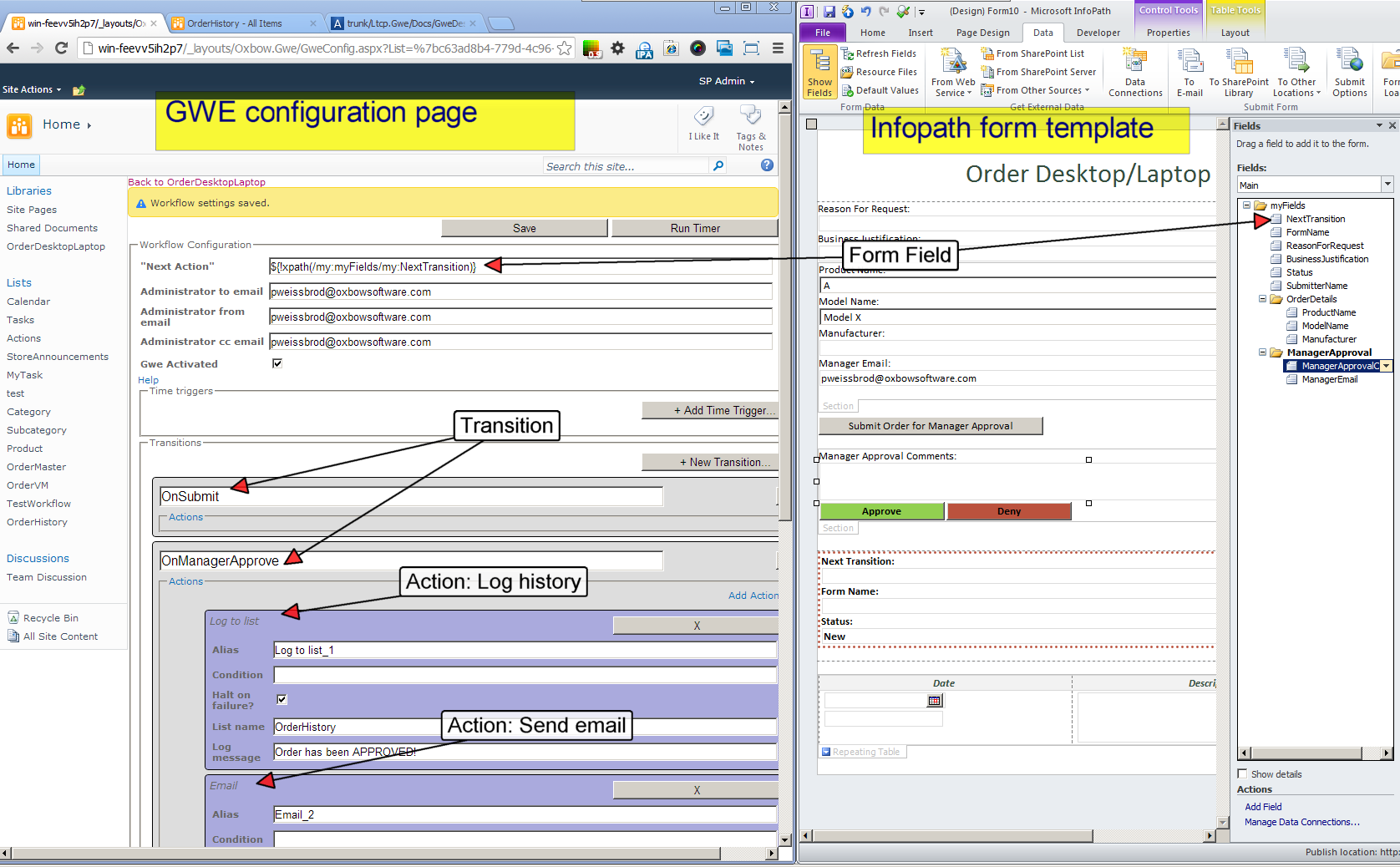


* The template has the static text “Good day” with a code block mixed in
* The code block is always wrapped in a ${ … } for GWE to recognize it as a code block
* Code blocks contain formulas that can be evaluated into text. Formulas are either functions or text.
* Functions always start with a “!” to signify a function. Functions have a name and contain 0 or more formulas as arguments. In the example above, the function name is listItemField and it has one argument which is text. This function will attempt to read the “UserName” field from the list item and return its value.
* Since a function is a formula and functions accept formulas as arguments, functions can be chained together, for example: [Good day ${!resolveuserfullname(!listItemField(“UserName”))}] will take the result of !listItemField(“UserName”) and attempt to resolve/return the full name.

The full list of functions available include:

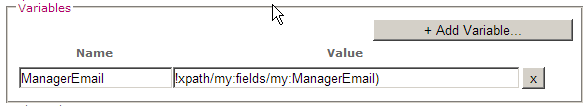
|  |  |
| --- | --- |
| **Function** | **Description** |
| !getmanagerforuser([username],[domain]) | returns the login name of the manager of the supplied identity. Requires User Prfolie features, will not work on SharePoint foundation. |
| !resolveuserfullname([username],[domain]) | returns the full name of the of the supplied identity. |
| !resolveuseremail([username],[domain]) | returns the email address of the of the supplied identity. |
| !xpath([expr]) | parses the form and evaluates the xpath expression |
| !getclientformurl() | returns the direct url of the list item |
| !getbrowserformurl([url]) | returns the url to view the list item in the web browser. The url parameter specified where to redirect after the form is closed. If the url parameter ="" then it will leave a blank screen. |
| !if([x1],[x2],[x3]) | if x1=true or 1 then return x2. if x1=false or 0 then return x3. if neither are true this is an error. |
| !and([x1]...[xn]) | Logical AND. returns true if all operands evaluate to true and/or 1. Requires at least one operand. Employs lazy evaluation. |
| !or([x1]...[xn]) | Logical OR. returns true if any operand evaluate to true and/or 1. Requires at least one operand. Employs lazy evaluation. |
| !listitemfield([fieldnameexpr]) | returns the value from the list item given an expression that evaluates a valid field name. |
| !iscontenttype([contenttypename]) | returns TRUE if the list item is associated with a content type matching the supplied name. |
| !equals([a],[b]) | returns TRUE if the value of expression a equals value of expression b. (trimmed and case insensitive comparison) |
| !contains([a],[b]) | returns TRUE if the value of expression a contains value of expression b. (trimmed and case insensitive comparison) |
| !concatenate([x1]...[xn]) | concatenates the expressions together |
| !currentsiteurl() | returns the site url containing the item (with the slash at the end eg http://mysite/) |
| !currentweburl() | returns the web url containing the item (with the slash at the end eg <http://mysite/myweb/>) |
| !now() | Returns the date and time in a format of yyyy-mm-ddTThh:mm:ss which matches infopath date/time type schema |
| !today() | Returns the date in a format of yyyy-mm-dd which matches infopath date type schema |
| !htmlencode([x]) | The equivalent of calling HttpUtility.HtmlEncode(x) |
| !htmldecode([x]) | The equivalent of calling HttpUtility.HtmlDecode(x) |
| !notequals([x],[y]) | The opposite of !equals([x],[y]) |
| !add([x],[y]) | Attempts to perform math add operation on arguments. The arguments must be both parseable as whole or floating point numbers. |
| !adddate([date],[amount],[scope]) | Use this for performing date/time adjustments: [date] is a value that can be parsed into a DateTime, [amount] is a value that can be parsed into an integer (negative or positive), and scope must be “hours”, “minutes”, “seconds”, “days”. This will return a date and time in a format of yyyy-mm-ddTThh:mm:ss which matches infopath date/time type schema |
| !evalxpath([expr]) | Evaluates the supplied xpath expression. Any expression used in infopath can be also evaluated with this function. Note that expressions are limited to the main data source and not secondary data sources. |
| !var([variablename]) | Returns the value of a global variable based on its name. |

# GWE Configuration Screen Example

The screenshot below illustrates the GWE configuration page for a form library on the left, and the form template design on the right. A field within the form template dictates which transition should be performed by GWE. When the user clicks a button, the rules for the button fill in the field with the correct transition and submit/close the form. Once the form is submitted, GWE is triggered, the transition is matched up with the form field, and all of the actions inside the transition are executed sequentially. 

# Workflow Variables: How they are defined and when to use them

GWE allows you to define multiple “variables” that can be referenced anywhere within the scope of the workflow. A variable has a name and a value. You can use this feature to prevent redundant code; replace multiple snippets of duplicate code with a variable. For example, say you frequently need to reference a field in an infopath form called “ManagerEmail” with the expression !xpath(/my:Fields/my:ManagerEmail). Define a variable with the name, set the value, and then you can reference this value as !var(ManagerEmail) multiple times while reusing the same value once.



# The GWE Actions

|  |  |
| --- | --- |
|  | Log to List:  When you need to log workflow activity, you can create a custom list with only a title and description field. Then for each transition you can add one of these actions. Specify the list name and the message and when the action executes a row will be inserted into the history list with the title being the name of the current list item and the description being the log message. You can report on data in the history list later. See the GWE web services as an example of history list reporting.  The list name and the message can be template with code blocks. |
|  | Form editor:  Use an action to update multiple fields inside an infopath form. Specify the target XPath to find the field and the new value.  The xpath and values can be template with code blocks. |
|  | List editor:  Same as form editor but works on list item fields instead of infopath fields |
|  | Send email:  This one is pretty self-explainatory. Every field can be template with code blocks. |
|  | Permission set:  This action changes item-level permissions |
|  | Custom code:  Need to do something unsupported? Here is the hook to write your own action. Write a class the implements the Oxbow.Gwe.Core.Contracts.IWorkflowTransitionCustomAction interface and deploy it in a DLL to the farm. Reference the fully qualified type name and assembly name and it will execute in the same fashion as the built-in actions |
|  | Inject XML:  Allows you to add XML into an infopath form as a workflow action. You would use this when you want to add an optional section, append and element, or add a row to a repeating table.  Supply the target xpath – this is an expression that must evaluate to an xpath for the parent element to insert the XML underneath. If the parent element is not found, this will be an error.  Append/prepend XML – The parent element may already have one or more child elements. Using append/prepend, you have the ability to inset at the beginning or end of the child element collection.  Xml – this is templated raw XML. In the sample to the left, we are inserting a new item element with a child element named “Id” with a value equal to the current item count plus one, and a child element named “Name” with a value equal to a workflow variable named “ManagerName” |

# Importing/Exporting/Deploying/Backing Up Workflows

Workflow configuration is saved as XML in the site property bag. You can export the workflow configuration to text by copying the contents of the export section at the bottom of the configuration page. You can import workflow configurations by overwriting this text with a different workflow and clicking import. This import/export procedure is the recommended practice for deploying workflows to different farms. When you need to make a backup of your workflow configuration, save a text file of the configuration in external backup storage.

# GWE SOAP Web Services

After installing GWE to SharePoint, a new ASMX web service is added to the web application under the URL <path\_to\_your\_site>/\_vti\_bin/GweWebServices.asmx. The web service contains helper methods you may find useful while developing an infopath form template.

|  |  |
| --- | --- |
|  | * GetFormHistory – accepts a “Form History” list name and an item name. If you have a list with two fields, “Title” and “Description” this method will read the list and return the “Description” values from each list item where the “Title” matches the item name. When you pair this with the “Log to History” list in the GWE event receiver you have the ability to show a full activity log of any list item. * GetManager – accepts a user name and returns the user’s manager name based on their sharepoint user profile. * GetUserEmail – Given a logon user name, get the user’s email * GetUserFullIdentity – Given a logon user name, get the user’s logon name, email and full name * IsUserMemberOfSharePointGroup – Given a logon user name and a sharepoint group name, check to see if the user is a member of the group. This is helpful when you want to show or hide sections of a form depending on the current user’s group membership. * GetSharePointUsersByGroup – Given a SharePoint group name, get the names and emails of all members of the group. This can be helpful when you need to populate a dropdown list box based on members of a group. |

# Developer Reference

## Prerequisites

* Visual studio 2010 or greater
* C#
* F#
* F# power pack v2 or greater

## Project structure

|  |  |
| --- | --- |
|  | Core   * Configuration – contains all system wide settings, the email template for error messages sent to the administrator, and a simple IOC container * Contracts – all system interface definitions * Models – The data structure for the WorkflowConfiguration * TemplateEngine – Contains all the function definitions. If you want to add a new function to the GWE modify the TemplateEngine * Utils – helper classes encapsulating peripheral concerns * WebServiceModels – Request and response models exchanged via the web service * WebServices – contains the logic behind the GWE web service * WorkflowEngine   + GweAgentRunner – Iterator pattern for traversing a farm and performing an action on list items where applicable.   + TimeTriggerJobRunner – the logic for the time trigger   + WorkflowEventReceiver – the event receiver * WorkflowTransitionActions – Each class represents the logic for each action handled by GWE   + WorkflowTransitionActionFactory – factory for instantiating the correct action given an action type |
|  | Lang project contains the lexer and the parser generator logic for the templating engine. It is a simple implementation of lex/yacc that accepts a string and returns an abstract syntax tree |
|  | SharePoint project contains the GWE workflow configuration page, event receivers and user controls for each kind of action. |
|  | The remaining projects are for unit tests and legacy upgrades. |

## How to make a custom action

1. Make a visual studio class library project.
2. Reference the Oxbow.Gwe.Core dll
3. Make a class that implements Oxbow.Gwe.Core.Contracts.IWorkflowTransitionCustomAction, for example:

**public class Test : IWorkflowTransitionCustomAction**

**{**

**#region IWorkflowTransitionCustomAction Members**

**public void Execute(SPListItem listItem) { ResolveType.Instance.Of<ILogger>().Info("Custom action has successfully received an item " + listItem.Url); }**

**#endregion**

**}**

1. Write the code for the action inside the execute method.
2. Sign the class library with a strong name key so you can deploy it to the GAC
3. Unit test it
4. Build the DLL and deploy it into the GAC via a sharepoint solution or some other vehicle
5. Inside GWE configuration screen, add a custom code action where you need it
6. Specify the fully qualified type name and assembly name in the configuration. If it should fail, you will receive an email like any other GWE action