Vignette Professional Services

StoryServer 5.0 Browser-Based Workflow

Content Production V-Paper

Version 1.0

June 14, 2000

External distribution is limited to customers who have participated in a VPS Content Production Workshop.

Trademarks

Author: Andrew Miller
Printed on 06/14/00 8:14 AM
Contact Vignette: 901 South MoPac Expressway, Building III, Austin, TX 78746-5776 or 512-306-4300.

Table of Contents

Introduction

You'll probably want to implement a browser-based workflow for most Vignette applications that need workflow because:

- Browsers are everywhere and users won't need to install any additional software, such as the StoryServer client tools, to access the workflow functionality.
- Browser-based workflow can limit the information presented to users. You can simplify the presentation and prevent accidental or malevolent access to data that users don't have privileges to see.

You can develop a basic browser-based workflow application using StoryServer and CMS API commands. Information and code samples show you how to integrate the workflow-related commands with the rest of the content management application (CMA). You'll see:

- Basic StoryServer 5.0 Tcl commands and the CMS API commands to use for creating and processing workflow in a StoryServer application. These commands are the basic building blocks from which workflow is constructed.
- Typical workflow tasks with code samples that highlight one approach to implementing the task. If a specific code sample accomplishes the task you need, you can use it as is, but it's better to develop your own function because Vignette doesn't support these code samples.
- Complex workflow tasks that are either very difficult to implement with the built-in StoryServer workflow support, or are not possible. The latter will require a custom solution.

Note that a highly complex browser-based workflow solution, using custom database tables and logic, is discussed in the *Custom Workflow* V-paper also available on the Knowledge Center.

Workflow Building Blocks

In addition to this review of workflow-related functionality, see the documentation listed in the Summary and References on page 8.

Command	Brief Description
RECORD AUTHORIZED	Performs a user authorization check
RECORD DELETE	Deletes a record
RECORD EXISTS	Determines if a record exists
RECORD UPDATE	Creates or updates a record
CMS CREATE_PROGRAM_TASK	Creates a program task
CMS CREATE_USER_TASK	Creates a user task

CMS LIST_USER_TASKS	Lists all tasks for a user and project
CONTENT_ITEM WF_COUNT	Gets the number of workflow tasks for item
CONTENT_ITEM WF_DELETE	Deletes a workflow task for an item
CONTENT_ITEM WF_GET	Gets workflow task attributes
CONTENT_ITEM WF_INSERT	Inserts a workflow task
CONTENT_ITEM WF_SET	Replaces workflow task attributes
USER_TASK START	Starts a task
USER_TASK FINISH	Finishes a task
USER_TASK REASSIGN	Reassigns a task

Note that a `RECORD` is a subclass of `CONTENT_ITEM`, so any content item commands can also be performed on `RECORD` instances.

Creating a Workflow

The easiest way to create a new workflow is to use the interactive support provided by the Project Manager in the Production Center. Following are some general tips:

- You have the option of creating a default workflow for templates, records, and files for a given project.
- Alternatively, you can create a unique workflow for each content type.

It really doesn't matter which alternative you choose since you should have a separate project for records.

Note that you can programmatically create a workflow with branches (forks), but you can't use the Project Manager for the branches. Create the workflow with the most likely branch. Then programmatically create the alternate branch based on information provided on page 6.

Processing a Workflow

Typical processing steps associated with workflow tasks include:

- Starting a task within the context of an edit session on page 3.
- Finishing a task within the context of an edit session and moving to the next task (approve) on page 4.
- Finishing a task and restarting workflow (reject) on page 4.
- Finishing a task and moving back to the previous task (reject) on page 5.
- Alternative workflow (branching) based on the outcome of the current workflow step on page 6.

In the examples that follow, an editor accesses an article to review as a workflow task through a browser interface. Specifically, the Task View of the Production Center is not used to manage workflow.

Starting a Task

A typical workflow task is for an editor to review an article that a writer has finished. In this scenario, the editor reviews the article in an edit form and starts the task. The editor makes changes, saves the changes, and completes the task.

First the editor needs to get a list of articles waiting for review:

```
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# Get the object ID of the project containing article records]
[SET projectID [theCMS LOOKUP_PROJECT ">Articles>News"]]
[# Retrieve the tasks for the editors group for all news
articles]
[SET result [theCMS LIST_USER_TASKS TABLE T INTO D WHO editors
PROJECT $projectID]]
<B>News Articles for Review</B><BR>
[# display list of article titles with link to edit]
[RETAIN taskObjId recordObjId]
[FOREACH ROW IN $D {
      [SET taskObjId [FIELD ObjectID]]
      [SET recordObjId [FIELD ContentItem]]
      [# get the record associated with the task]
      [SET result [theCMS RECORD theRecord [SHOW recordObjId]]]
      [# fetch the title of the article]
      [SET articleId [FIELD theRecord.RecordID]
      [SEARCH TABLE article INTO articleList SQL "select title
            from [FIELD theRecord.NameOfTable] where [FIELD
            theRecord.PrimaryKeyName] = [SHOW articleId]]
      [CURL /article/edit [SHOW articleId] [FIELD title [FIRST
            articleList]] SAVE_STATE]
}]
```

If the name of the StoryServer record is the same as the article title, then you can eliminate the `SEARCH TABLE` command in the preceding example. Also note that each link displays the article title and passes the following:

- The primary key of the article table in the `CURL ID`.
- The object ID of the task and StoryServer record in an appended query string.

In the following code sample, clicking on an article:

- Fetches the contents of the article and displays it in a form.
- Starts the workflow for the task.

```
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# get the task associated with this content item]
[SET result [theCMS USER_TASK theTask $taskObjId]]
[# start the task]
[SET result [theTask START]]
      .
      . fetch article contents and display in a form
      .
```

Finishing a Task

In this scenario, the editor reviews the article, makes any corrections or updates, and saves the article. At this time the task should be finished, so the next task in the workflow sequence is started:

```
      .
      . save article updates
      .
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# get the task associated with this content item]
[SET result [theCMS USER_TASK theTask $taskObjId]]
[# finish the task]
[SET result [theTask FINISH]]
```

Finishing a Task and Restarting Workflow

In this scenario, the article requires more work from the writer, such as more research or major revisions that the editor doesn't do. The editor wants to return the article to the writer and have the code restart the workflow process from the beginning.

The easiest way to restart workflow from the beginning is to delete and then recreate the StoryServer record associated with the article:

```
      .
      . attach general comments about what needs to be corrected
      .
[# projMgmtId is the management id of the project containing the
record]
[# delete original record]
[RECORD DELETE [SHOW ID] article "" ""]
[# recreate record, restarting workflow in the process]
[RECORD UPDATE [SHOW ID] article "" "" id [SHOW projMgmtId]
[SHOW articleTitle]]
```

Finishing a Task and Moving to the Previous Task

The preceding scenario shows that it is fairly simple to restart workflow *from the beginning* if the current task does not complete successfully. However, moving to the *previous task* if the current task does not complete successfully is a more challenging coding task.

The workflow restarts from the beginning but skips enough steps to reach the task before the current (unsuccessful) task:

```
[# It is assumed that the record Object ID is known]
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# get the record]
[SET result [theCMS RECORD theRecord $recordObjId]]
[SET recWfCount [theRecord WF_COUNT RECORDS]]
[SET result [theCMS PROJECT theProject [FIELD
theRecord.Project]]]
[SET projWfCount [theProject WF_COUNT RECORDS]]
[SET procCount [expr $projWfCount - $recWfCount]]
[
if {$procCount > 1} {
      # restart workflow
      RECORD DELETE [SHOW ID] article "" ""
      # recreate record, restarting workflow in the process
      set bundle(Name) $articleTitle
      set bundle(Project)$projObjId
      set bundle(NameOfTable) "article"
      set(PrimaryKeyName) "id"
      # Create the record.
      set recordObjId [theCMS CREATE_RECORD bundle]
      set result [theCMS RECORD theRecord $recordObjId]
      for {set i 2} {$i < $procCount} {incr i} {
            # finish first number of tasks to catch up to task
            # prior to this one
            set result [theCMS USER_TASK theTask [FIELD
                  theRecord.CurrentTask]
            set result [theTask FINISH]
      }
}
]
```

The preceding code sample compares the number of tasks remaining for the record to the number of tasks defined for records in the project. This difference is used to determine how many tasks to skip when recreating the workflow so that the task previous to the current task becomes the new current task.

For example, if five tasks have been defined for records in a project and the current task for a record is task #4, then one task is left in the workflow for the record. If the workflow is restarted for the record and the first two tasks are skipped, then the current task will now be task #3, just before task #4.

Alternative Workflow Task

In this scenario, you can alter subsequent workflow tasks based on the outcome of the current task. For example, an editor may approve the content of an article but may want a publisher to further review the article to ensure that it is consistent with the overall mission of the publication.

Since this additional task is not part of the normal workflow process, it will not be included in the default workflow. You can add this task programmatically:

```
      .
      . save article updates
      .
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# get the record associated with this content item]
[SET result [theCMS RECORD theRecord $recordObjId]]
[# insert a new workflow task as the next task]
[SET bundle(Name) “Publisher Review”]
[SET bundle(What) $articleTitle]
[SET bundle(Who) publisher]
[SET result [theRecord WF_INSERT bundle]]
[# get the current task associated with this content item]
[SET result [theCMS USER_TASK theTask $taskObjId]]
[# finish the task]
[SET result [theTask FINISH]]
```

Retrieving Workflow Status

A manager frequently wants to retrieve status information about workflow tasks. For example, to understand the workload balance among employees, the manager needs to:

- View a list of tasks assigned to a particular user.
- View a list of tasks that have been assigned but not started within a given period of time.

Viewing User Tasks

Getting a list of tasks currently assigned to a specific user is straightforward:

```
[# Connect to the CMS.]
[SET result [CMS theCMS]]
[# Get the object ID of the project containing article records]
[SET projectID [theCMS LOOKUP_PROJECT “>Articles>News”]]
[# Retrieve the tasks for the editors group for all news
articles]
[SET result [theCMS LIST_USER_TASKS TABLE T INTO D WHO editors
PROJECT $projectID]]
Group editors has [llength [SHOW D]] tasks.
```

This is the same code as in the scenario on getting a list of articles to review. In that example, the goal was to provide an easy interface for an editor to access the article and automatically start a task. In this example, however, the goal is to report the number of tasks assigned to a particular group or individual.

Viewing Late Tasks

One important management responsibility is to track employees' ability to process assigned tasks in a reasonable time. You can extract this information but only by accessing the task table (`vgn_tk`) in the system database. Note that the template execution environment must have `SEARCH` permission for the task table.

The following example reports a list of tasks that have been assigned and are more than 48 hours old:

```
[# find any tasks older than 48 hours]
[SET oldTime [DATE_FORMAT 48 "H" INTERVAL]]
[SET nowTime [DATE_FORMAT NOW DATETIME]
[# get a list of all Assigned tasks]
[SEARCH TABLE tk INTO tkList SQL "select bobdb__objval from
ss_system.vgn_tk where State = 'Assigned'"]
[FOREACH ROW IN [SHOW tkList] {
      [SET raw [FIELD bobdb__objval] NOEVAL]
      [# extract the DATETIME string of when the task was
            created]
      [regexp {CreateTime,D([^\]]*)} $raw dummy cTime; NULL]
      [SET crTime [DATE_FORMAT $cTime DATETIME]]
      [IF [DATE_COMPUTE [DATE_COMPUTE $nowTime – $crTime] >
            $oldTime} {
            [FIELD Name] [FIELD Who]<BR>
      }]
}]
```

Because the preceding code sample relies on direct access to one of the StoryServer system tables, which are not documented and are subject to change with each new release of StoryServer, you should not make this approach your first choice.

Limitations of StoryServer Workflow Support

StoryServer provides workflow support to handle most of the basic workflow scenarios. While this functionality is suitable for the majority of situations, there are other situations where the built-in support is lacking and a custom workflow solution is required. These limitations include:

- It is difficult to go back to a previous step in the workflow. You can go to the previous step programmatically, but the support is not inherent in the StoryServer platform.
- It is difficult to determine when a task was assigned to a user or group. You must extract this information directly from the StoryServer system tables.
- Branches must be handled programmatically.

- Joins are not possible.
- The workflow history provides only basic information – the person who completed a task and the time. You can't attach any comments to a task.

If the workflow requirements are not constrained by these limitations, then the built-in workflow is adequate. Otherwise, you need to develop a custom workflow solution.

Summary and References

For straightforward workflow management, the support provided with the StoryServer and CMS API commands is adequate. Information and code samples are provided to help you manage and process basic workflow tasks. For more complex workflow requirements, create a custom solution as discussed in the *Custom Workflow* V-Paper.

The following StoryServer 5.0 documents are useful for understanding workflow concepts and for details about using the workflow-related commands:

- *Template Command Reference*
- *CMS API Template Command Programmer's Reference*
- *Production Center Guide*

Appendix A. Document Control

Document Revision History

Version	Release Date	Revised By	Revision Description
0.1	06/08/2000	Andrew Miller	Initial distribution
0.2	06/12/2000	Jody Kelly	Initial editing/rewriting
0.3	06/14/2000	Andrew Miller	Final technical review
1.0	06/14/2000	Jody Kelly	Readied for distribution

Document Storage

This document was created using Microsoft® Word 97. The file is stored in the Knowledge Center on the Specialized Services Downloads page.

Document Owner

Andrew Miller is responsible for developing this document. Jody Kelly is responsible for maintaining it.