INTERWOVEN

TeamSite®

Workflow Developer's Guide

Release 5.5.2

INTERWOVEN

Interwoven, Inc.
803 11th Ave.
Sunnyvale, CA 94089

http://www.interwoven.com

Printed in the United States of America
Part # 40-00-10-11-00-552-210

Table of Contents

About This Book

The *Workflow Developer's Guide* is a guide to installing, configuring, and using WorkflowBuilder. Additionally, it describes the files used by TeamSite workflow and how to create and edit them without WorkflowBuilder. This document is primarily intended for TeamSite Administrators and Master users, and workflow developers.

Notation Conventions

This manual uses the following notation conventions:

Convention	Definition and Usage
Bold	Text that appears in a GUI element (including menu items, buttons, or elements of a dialog box) and command names are shown in bold. For example: Click **Edit File** in the Button Bar.
Italic	Book titles appear in italics. Terms are italicized the first time they are introduced. Important information may be italicized for emphasis.
`Monospaced`	Commands, command-line output, and file names are in monospaced type. For example: The `iwextattr` command-line tool allows you to set and look up extended attributes on a file.
`Monospaced italic`	Monospaced italics are used for command-line variables. For example: `iwckrole` *`role user`* Means you must replace *`role`* and *`user`* with actual role and user values.

Convention	Definition and Usage
`Monospaced bold`	Monospaced bold represents user input. The > character that appears before a line of user input represents the command prompt and should not be typed. For example: **`>iwextattr -s project=proj1 //IWSERVER/default/main/dev/WORKAREA/andre/products/index.html`**
`Monospaced bold italic`	Monospaced bold italic text is used to indicate a variable in user input. For example: **`>iwextattr -s project=`*`projectname workareavpath`*** means that you must insert the values of *`projectname`* and *`workareavpath`* when you enter this command.
`[]`	Square brackets surrounding a command-line argument mean that the argument is optional.
`|`	Vertical bars separating command-line arguments mean that only one of the arguments can be used.

Notation of iw-home on UNIX and Windows Systems

This manual does not use the UNIX notation (`iw-home`; note the lack of italics) except when specifically referring to procedures performed only in UNIX.

This manual uses the Windows version of `iw-home` notation (italicized *`iw-home`*) when discussing both UNIX and Windows systems. The italics are an Interwoven convention identifying *`iw-home`* as a variable. You should interpret the *`iw-home`* notation used in this manual as follows:

- On UNIX systems, `iw-home` is the literal name of the directory containing the TeamSite program files.
- On Windows systems, *`iw-home`* is the symbolic name of the directory that contains your TeamSite program files. The default value of *`iw-home`* on Windows systems is:

  ```
  C:\Program Files\Interwoven\TeamSite
  ```

  The administrator performing Windows installation may have chosen an installation directory different from the default.

Windows Path Name Conventions

In most cases, you can specify path names using standard Windows naming conventions (which allow you to include spaces in path names). However, in some situations it might be necessary to use MS-DOS naming conventions, which stipulate that no single file or directory name in a path can contain a space or more than eight characters. If you encounter unexpected system behavior after entering a path name using Windows NT naming conventions, enter the path name again using MS-DOS conventions.

For example, instead of:

```
> C:\Program Files\Interwoven
```

you can try:

```
> C:\Progra~1\Interw~1
```

You can use the `dir /x` command to display the long and short versions of the file names in the current directory.

Online Documentation Errata

Additions and corrections to this document are available in PDF format at the following website:

`http://support.interwoven.com`

When you reach this site:

1. Click **Download**.
2. Enter your user name and password.
3. Click **All Documentation**.
4. Click **Current Release Notes**.
5. Click the link to the appropriate PDF file.

Chapter 1

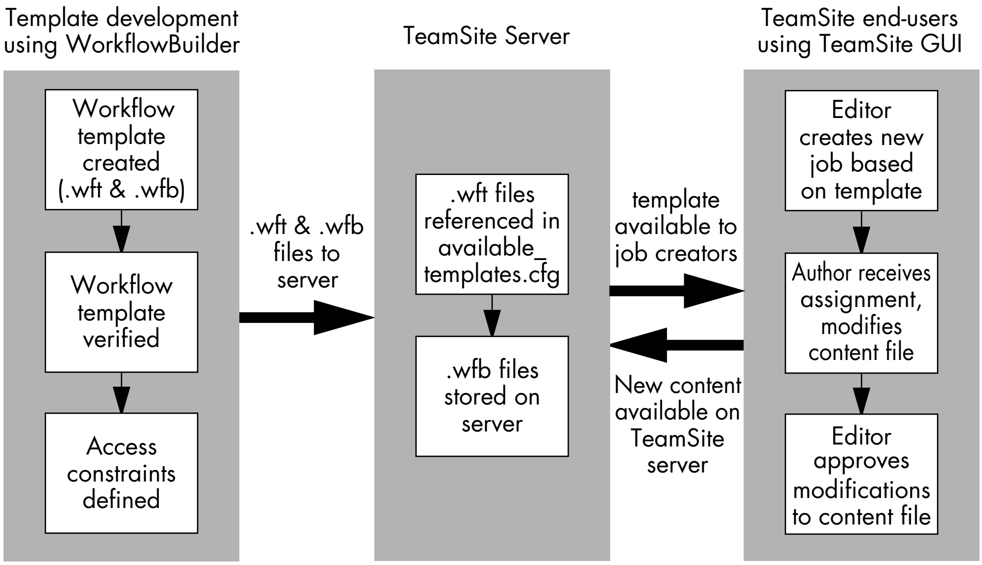
Introduction

Workflow encompasses the procedures, tasks, people, and equipment that define business practices within an organization. Using TeamSite to define and automate workflow ensures that the business practices associated with your web-based assets are performed in a logical and consistent manner leading to better organization and increased productivity.

TeamSite's workflow system consists of three main components:

- **WorkflowBuilder**—Client-side application that enables workflow developers to build workflow templates using an intuitive, drag-and-drop graphical interface which can then be transferred to a TeamSite server. WorkflowBuilder is supported on Windows NT and Windows 2000 platforms.
- **WorkflowBuilder Server**—Working with your TeamSite server, it provides a framework for controlling complex website production processes. Your custom workflow templates and sample templates (included with WorkflowBuilder) are stored on the TeamSite server and made available to end users. WorkflowBuilder Server is supported on Windows NT, Windows 2000, and Solaris platforms.
- **Browser-based client-side user interface**—Displays forms that enable end users to enter data that defines and controls specific workflow actions. These end users include those creating jobs based on the workflow templates, and those performing the tasks contained within these jobs.

These three components are depicted in the following illustration.



Simplified Workflow Template Lifecycle

What's New in TeamSite 5.5.2 Workflow?

TeamSite 5.5.2 includes a new group of workflows known as *solution* workflows. These workflows were designed to incorporate functionality that was commonly being implemented either by modifying the Interwoven-provided workflow template files (`wft`), or by modifying the Perl code.

Three of these workflows are variations on the `author_submit` default workflow included with earlier TeamSite releases. The other three new workflows are configurable alternatives to the previous default workflows included with earlier versions of TeamSite.

The solutions workflows are described in Chapter 8.

In addition, if you are upgrading from pre-5.5 releases of TeamSite, you should also be aware of the following recent advancements to TeamSite workflow that may have been implemented since you installed your version:

- **Dynamic Attributes**—The ability to change properties of instantiated jobs and tasks, including the ability to:
  - Change the owner of a job or a task
  - Add and remove users and groups from a grouptask
  - Modify the timeout period for a task
  - Modify the areavpath of a task
  - Modify various task attributes (for example, lock or read-only)
- **Nested Workflow**—The ability to nest any number of workflows to create larger, more complex processes. Nested workflow can be initiated by:
  - Pre-configured job specification files
  - Users in the course of a job using the standard WFT instantiator

WorkflowBuilder 5.5 includes the following features:

- **Additional Sample Workflow Templates**—For demonstrating the functionality of timeout notification of editors and nested jobs.
- **Workflow Constraints**—Teamsite enables you to control access to specific workflow templates by setting constraints on the template files when they are published to the TeamSite server. The constraints can be set by workflow type, users, roles, and branch.
- **Template Titles**—Workflow template files can be assigned a title (which differs from the actual file name) when published to the TeamSite server.
- **Template Validation**—Verify Templates feature ensures the workflow template has a Start Task, an End Task, the proper transition types between tasks, and a job owner
- **Attach Files to Jobs**—Workflow templates can specify files that are attached to tasks that are ultimately assigned to authors.
- **Printable workflow template files**—WorkflowBuilder now includes the ability to print and preview workflow templates.

- **Perl Code window in WorkflowBuilder**—Enables template developers to add custom Perl code to create custom variables.
- **Remote template management**—WorkflowBuilder now includes functionality that enables you to copy, delete, and change the status of workflow templates already transferred to your TeamSite server.

Workflow Terminology

This section defines workflow terminology as it relates to TeamSite. Note that many of these terms have more general definitions outside of the context of TeamSite.

Tasks

A *task* is a unit of work performed by a single user or process. Each task is associated with a TeamSite branch and workarea and one or more files. The user or process owning a task can modify, add files to, or remove files from the task (provided the task is not a read-only task for content approval).

See "Tasks" on page 19 for a detailed description of the 11 task types defined within WorkflowBuilder.

Additionally, tasks have two possible states: active and inactive. A task becomes active when its predecessor task signals it to do so (predecessor tasks and conditions for activation are all configured as part of the workflow model). After the task has been activated, users or external programs can work on it. For example, after a user task has been activated, the user can work on the files contained in the task. After an external task has been activated, the appropriate external program can run on the files contained in the task. Inactive tasks are tasks that have been completed, or that have not been activated yet.

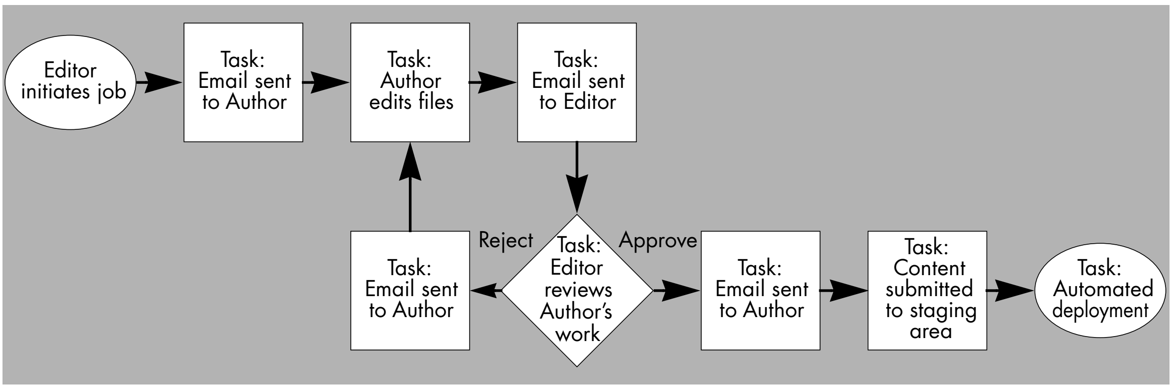
Note: A workflow task cannot have more than 512 predecessors.

Workflow Models

A *workflow model* is a general workflow specification that can be used repeatedly. Each workflow model describes a process that can include user tasks and a wide variety of automated tasks. Workflow models typically are designed by business managers and configured by a system administrator or Interwoven Client Services (`http://www.interwoven.com/services`).

The following diagram shows a simple assign-edit-approve workflow model. Email is sent to the participants at each stage of the process, and an automated task is performed at the end.

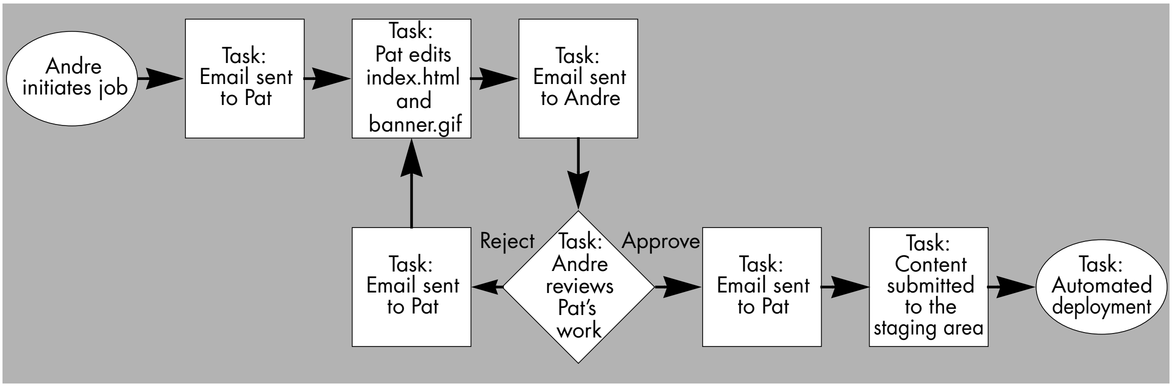
Note that the people involved are not actual people, but are represented as an editor and an author, also note that no specific files are mentioned. This is an important distinction between the generalized workflow model and the job instance described in the next section.



Jobs

A *job* is a specific instance of a workflow model containing a set of interdependent tasks. One example of a TeamSite job is the set of tasks needed to prepare a new section in a marketing website to support a new product launch.

In TeamSite, a description of a workflow model is called a *job specification*. When a job specification is loaded into the workflow subsystem it becomes a *job instance*. Each job is a specific instance of a workflow model. When a job is created, the job creator must supply all the specific information for that job. For example, the workflow model depicted in the previous section could be used for the marketing web site's new product launch. Note that the job specification (shown in the following diagram) includes specific TeamSite users: Andre (the editor) and Pat (the author), and specific files that need to be edited: `index.html` and `banner.gif`.



Because jobs follow predefined workflow models, tasks cannot be added to or removed from individual jobs.

Workflow Templates

Workflow templates are XML files that describe a particular workflow model. You can create these files using WorkflowBuilder, then transfer them to your TeamSite server where they can be used to create a new job.

Note: The term "workflow template" is sometimes used to describe either a file that describes a workflow model, or a file that describes a particular job instance (also known as a *job specification file*). Both types of file end with the `.wft` extension.

Job Specification Files

Job specification files are XML files that describe a specific job. You can create these files using WorkflowBuilder, then transfer them to your TeamSite server where they are immediately invoked.

Workflow Elements

There are three major types of workflow elements defined in WorkflowBuilder and TeamSite:

- Tasks
- Transitions
- Conditions

These elements are introduced in the following sections.

Tasks

You can include the following tasks in your workflow template or job specification file. For more information about task attributes, see page 22.

Task	Description
User	Defines the task that appears on a user's task list.
Group	Appears in the task list of each member of the arbitrary group of users specified in the task. A group task becomes identical in behavior to a user task when one user from the group takes ownership of the task via the TeamSite GUI or the CLT `iwtaketask`.
Submit	Performs a submit operation on its contained files. If the submit task succeeds, the specified successor tasks are signaled. If the submit task fails, it goes into a special state that the user interface can detect. When the user interface resolves a conflict, it retires the operation so that the job can continue. For the purposes of workflow, a submit task is considered successful even if some of its contained files were not submitted because of not being up to date with the staging area.
Update	Performs a Get Latest or Copy to Area on its contained files. If the update task fails because of conflicts, it goes into a state like that for a failed submit task. The user interface is responsible for resolving conflicts and retrying the update task.
CGI	Behaves much like an external task, but does not run a `<command>` element.
External	Runs external programs when it is activated.
Email	Sends email to specified users.
Dummy	Used as a spacer or timed task. Dummy tasks let a workflow designer create a time interval unrelated to any actual job activity. A dummy task has no owner or areavpath.

Lock	Attempts to acquire locks on files it owns. If it succeeds, it transitions to the successors specified in its `success` transition. If it fails, it transitions to the successors specified in its `failure` transition. This provides users with a way of backing out of a job or choosing an alternate path in a job that cannot acquire its locks.
End	Ends a job.
Nested Workflow Job	Creates a task that is started when another task within a job reaches a certain state. Nested tasks are considered child tasks and the completed tasks that trigger them are considered parent tasks. The attributes are similar to a External tasks except you have to enter a workflow template path or a job file name instead of a command name.

Transitions

You can choose among four different kinds of transitions for the transition you place between all types of tasks, except the End task. An End task can only accept Successor or Timeout transitions from predecessor tasks; an End task ends a workflow and does not transition to any other task.

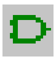
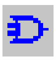
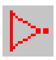
Transitions can be qualified with logical conditions, such as AND, OR, or NOT.

Transition	Description
Successor	The most common type of transition. A successor transition specifies the next task in the workflow.
Timeout	Sets an optional time limit for the completion of a task. When time runs out the task is inactivated and its successors are signalled to become active. The time value can be specified in one of two forms: the number of hours and minutes after the task becomes activated that the timeout should occur, or the month day, year, hour, and minute at which the timeout should occur.
Reset	Resets a task so that it is in its original state. Note the distinction from “inactivate” which maintains information about its prior state.
Inactivate	A task becomes inactive at the time it signals its successors. However it is sometimes necessary to inactivate tasks other than those which have signalled a task when that task becomes active.

Conditions

AND, OR, and NOT condition elements specify the conditions under which a task will become active. When a finishing task signals a successor task, the successor task notes that the finishing task has signaled and then evaluates the logic of the element to determine if it should become active.

For example, if you want Task C to be activated only when Task A and Task B have been completed, draw transition lines from the tasks A and B to a AND element, then draw a transition line from the AND element to Task C. You can qualify a transition with any one of the following condition elements:

Condition	Description
 AND	All tasks linked to this element must be completed to activate a successor task.
 OR	One of the tasks linked to this element must be completed to activate a successor task.
 NOT	The task must not be completed to activate a successor.

To add a conditional element to a transition:

1. Select a condition element from the toolbar.

   When you move the pointer to the canvas, a graphic icon appears under it to indicate that a mouse click will place a graphic on the canvas.

2. Draw transition lines from predecessor tasks to the condition, and from the condition to the successor task.

Task Attributes

Workflow elements (jobs, tasks, and transitions) all have special attributes which you must configure when you create a workflow template or job. Different types of tasks have different possible attributes. Some of these attributes are mandatory and some are optional.

Some attributes can be set using variables. If attributes are set with user variables, the file will be a workflow template which may be invoked through the **New Job** menu item or the **Submit** button in TeamSite. If attributes are not set with user variables, the file will describe a specific job.

The attributes that are available for different tasks and other workflow elements are described in the following sections.

CGI Task Attributes

These attributes are available for all CGI tasks:

Attribute	Description
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)
Immediate	When set to Yes (`immediate="t"`) this attribute specifies that any other cgitasks in the workflow by executed immediately.
Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be Yes or No). For more information, see the *TeamSite Administration Guide*.
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No, but there must be one task set to Yes).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Command	Specifies the full path of the program to be run on activation The program it references must be located in `*iw-home*/httpd/iw-bin/`. (Required entry.)

Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Dummy Task Attributes

These attributes are available for all dummy tasks:

Attribute	Description
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Dummy tasks require a timeout transition.

Email Task Attributes

These attributes are available for all email tasks:

Attribute	Description
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)

Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be true or false). For more information, see the *TeamSite Administration Guide*.
Retry	Specifies whether or not the email sent by this task should be resent if for any reason it is not delivered to the recipient (default is Yes).
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be true or false).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Email	Specifies the email address to send mail to. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

End Task Attributes

This attribute is available for all end tasks:

Attribute	Description
Name	(Required) Name of the task. Each task within a job must have a unique name.
Description	A description of what the task does.

External Task Attributes

These attributes are available for all external tasks:

Attribute	Description
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)

Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be Yes or No). For more information, see the *TeamSite Administration Guide*.
Retry	Specifies whether or not the command executed by this task should be retried if it fails (default is Yes).
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Command	Specifies the full path of the program to be run on activation, followed by any initial arguments. For more information, see the *TeamSite Administration Guide*. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Group Task Attributes

These attributes are available for all group tasks:

Attribute	**Description**
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be true or false). For more information, see the *TeamSite Administration Guide*.
Readonly	Specifies whether users can add, remove, or modify files in the task (can be true or false).

RetainOwner	After a group task has had someone take ownership of the first time, setting this to Yes causes the group task to behave similar to a user task by retaining the same owner from that point on. For example, if you loop back to an previous task and once again transition to that task, it will be considered as transitioning to the person who first claimed ownership of the task and *not* go back into a "shared by pool" waiting for someone to claim ownership again.
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be true or false).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Sharedby	Specifies the set of users who share this group task. These users can be individual TeamSite users, or Windows or UNIX user groups. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Lock Task Attributes

These attributes are available for all lock tasks:

Attribute	Description
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be true or false).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks. In the case of a workflow template, this can be a relative path under `iw-home/local/config/wft`.

Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Lock tasks also have two types of transition that are not available for other tasks: Success and Failure. Both type of transitions are required for all lock tasks.

Nested Job Task Attributes

These attributes are available for all nested tasks:

Attribute	**Description**
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
Wffile	Specifies the full path of the workflow template or job specification file to be run on activation. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Submit Task Attributes

These attributes are available for all submit tasks:

Attribute	**Description**
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.

Owner	Username of the task's owner. (Required entry.)
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
Skip Conflicts	Specifies whether or not to submit conflicting files (can be Yes or No).
Skip Locked	Specifies whether or not to submit locked files (can be Yes or No).
Override	Specifies whether or not to overwrite the staging area version of conflicting files (can be Yes or No).
Unlock	If set to Yes, then the submittask will unlock all files following successful submission.
SaveComments	If set to Yes, the comments are saved. Defaults is Yes.
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Update Task Attributes

These attributes are available for all update tasks:

Attribute	**Description**
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
Delete	Specifies whether or not to propagate deleted files to the destination area (can be Yes or No).
Overwrite	Specifies whether or not to overwrite the destination area version of conflicting files (can be Yes or No).

Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be Yes or No). For more information, see the *TeamSite Administration Guide*.
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
SrcAreaVPath	Specifies the area from which files are copied. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

User Task Attributes

These attributes are available for all user tasks:

Attribute	**Description**
Name	Name of the task. Each task within a job must have a unique name. (Required entry.)
Description	A description of what the task does.
Owner	Username of the task's owner. (Required entry.)
Lock	Specifies whether or not the task will try to acquire locks on the files it contains (can be Yes or No). For more information, see the *TeamSite Administration Guide*.
Readonly	Specifies whether users can add, remove, or modify files in the task (can be true or false).
Start	Specifies whether or not this task is activated at the time that the job is invoked (can be Yes or No).
AreaVPath	Specifies the TeamSite area associated with this task. (Required entry.)
Files	Specifies the files that the actions of a task affect. WorkflowBuilder can only specify files for start tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.

Workflow Attributes

These attributes are available for all jobs and workflow templates:

Attribute	Description
Name	Name of the job or workflow template. (Required entry.)
DebugMode	Sets the Debug flag to On in the workflow template. When the job is instantiated, rather than running the actual job, a Debug output page is created. It contains details about what the instantiated XML looks like plus what Perl variables were declared.
PreselectedFiles	Enables end-users (using the TeamSite GUI) to configure the workflow to automatically include files selected before starting the job. The preselected files are attached and sent to other users along with assigned tasks.
Variables	Key (variable name)/value pairs that are set at the Task or Job level for use by a CGI task or the TeamSite GUI. For example, the Priority variable is used at the GUI-level to set the priority of a task.
Description	A description of what the job does.
Owner	Username of the job's owner. (Required entry.)

Dynamic Attributes

TeamSite 5.5.2 includes the functionality to modify a number of important job and task attributes given certain restrictions (including, but not limited to, those listed in the following table). This ability to make changes to tasks already instantiated is supported from the command-line and from the TeamSite GUI.

Modification	Restriction
Change the owner of a job	Limited to masters and the current owner of a job
Add, remove, or change job variables	Limited to masters and the current owner of a job
Change the owner of a task	Limited to masters, job owner, and task owner
Add and remove users and groups from a grouptask	Limited to masters, job owner, and task owner (command-line only; not supported in GUI)
Change the timeout period for a task	Limited to masters, job owner, and task owner

Modification	Restriction
Change the areavpath of a task	Limited to masters, job owner, and task owner
Change various task attributes (for example: lock, read-only)	Limited to masters, job owner, and task owner
Add, remove, or change task variables	Limited to masters, job owner, and task owner (command-line only; not supported in GUI)

The procedures for making these dynamic modifications from the Job Administration GUI are contained in the next section. The command-line reference begins on page 34.

Dynamically Modifying Attributes Using the GUI

TeamSite includes a new Job Administration GUI for managing and modifying jobs and tasks, and their corresponding attributes. Complete the following procedure to display a job in the Job Administration GUI:

1. Log in to TeamSite by typing the following command in the Location (Netscape) or Address (Internet Explorer) field of your browser:

   `http://`*`TeamSite_server_name`*`/iw`

   The TeamSite login screen is displayed.

2. Select the Master role from the drop-down menu.

3. Enter your username and password in the corresponding fields.

4. Click **Login**.

   The main TeamSite window is displayed.

5. Select New Job from the File drop-down menu.

   The New Job window is displayed.

6. Complete the following steps in the New Job window:

   a. Select one of the templates, for example: Author Assignment.

   b. Type a description of the new job, for example: **`Test of Dynamic Attributes`**.

c. Click **New Job**.

The New Job Template window is displayed with the description you entered in step b in the Job Description field.

7. Complete the following steps in the New Job Template window:

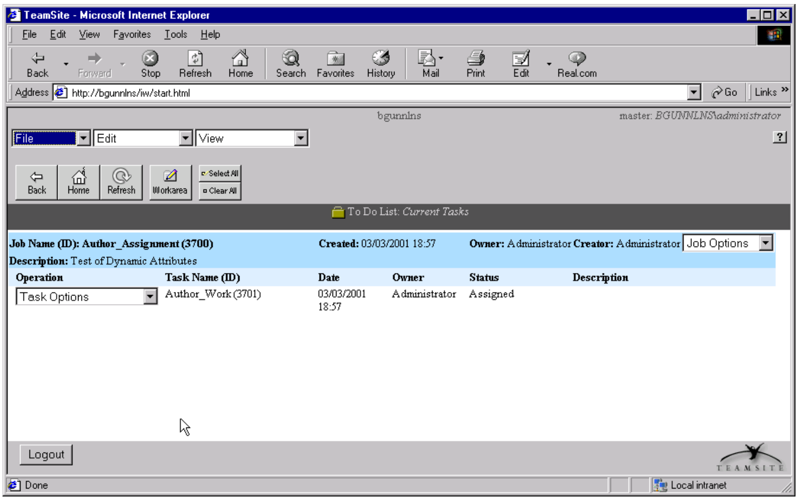
a. Select an Author from the drop-down menu.

b. Select a Branch from the drop-down menu.

c. Type the name of a workarea in the Enter Workarea field.

d. Click **Run Job**.

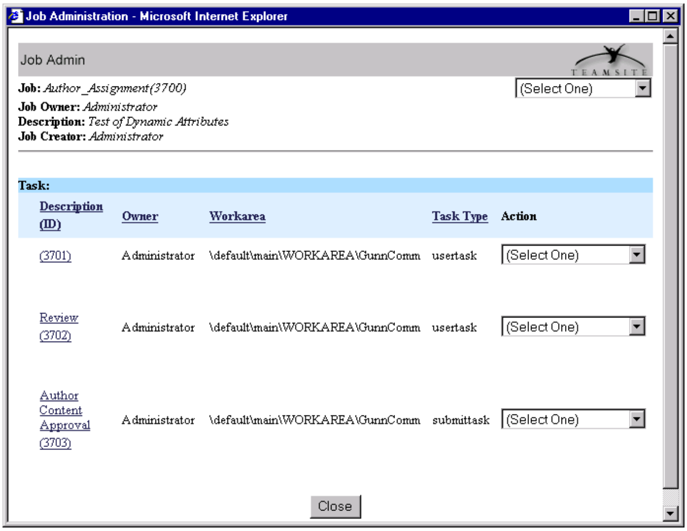
The main TeamSite window is displayed.



Note that the owner of the Job and the Task is the user "Administrator"

8. Select Job Admin from the Job Options drop-down menu.

The Job Admin window is displayed.



9. Select the attribute you want to change from one of the four drop-down menus.

   The top drop-down menu contains only the Change Job Owner option. The other three offer the following options:

   – Change Task Owner

   – Change Area

   – Change Attributes

10. To change an attribute, perform the procedure described in the following table:

Change	Window Displayed	Procedure	Result
Job Owner	Change Job Owner	• Enter the new job owner (this user must be a valid TeamSite user). • Click **Change**.	The Job Admin window displays the new Job Owner.
Task Owner	Change Task Owner	• Enter the new task owner (this user must be a valid TeamSite user). • Click **Change**.	The Job Admin window displays the new Task Owner.

Change	Window Displayed	Procedure	Result
Area	Change Task Area	• Enter the new workarea. • Click **Change**.	The Job Admin window displays the new Task Owner.
Attributes (on a task)	Change Task Attributes	• Click the corresponding True or False option button to define whether the corresponding task is locked or read-only. • Click **Change**.	The Job Admin window displays the new attributes.
Attributes (on author content approval)	Change Task Attributes	• Click the corresponding True or False option button to define the corresponding task: – skipconflicts – skiplocked – override – unlock – savecomments • Click **Change**.	The Job Admin window displays the new attributes.

Dynamically Modifying Task and Job Attributes from the Command-line

In addition to the GUI-based functionality, TeamSite 5.5.2 includes Command-Line Tools (CLTs) that provide equivalent functionality as the GUI version. These CLTs include:

- `iwsettaskattrib`
- `iwsettasktimeout`
- `iwrmtaskowner`
- `iwsetjobowner`
- `iwsettaskownerandarea`
- `iwaddtaskgroup`
- `iwsetjobdescription`
- `iwaddtaskowner`
- `iwrmtaskgroup`

Variables

Variables allow you to specify attributes that are subject to change. There are three types of variables in WorkflowBuilder:

- User Variables
- System Variables
- Custom Variables

You can use these variables when setting attributes of a workflow element. Each type of variable is described in the sections that follow.

User Variables

User variables are variables which will appear in a workflow template in TeamSite. The job creator can set these variables to describe a specific job.

For example, you might describe the Owner attribute of a user task with a user variable. When a job creator selects the workflow template in TeamSite, the task owner in the New Job form would have to be set.

System Variables

System variables are characteristics of the system, or of the user who is creating a job from your workflow template.

Available system variables are:

Variable	Description
`iw_home`	Location of the TeamSite home directory.
`iw_role`	Role of the user instantiating the job.
`iw_session`	Current session string.
`iw_template_file`	Path to the current workflow template.
`iw_template_name`	Name of the current workflow template.
`iw_use_default`	If all user variables have default values and this is enabled, those default values are used to create the job (the value entry form does not appear at job creation time).
`iw_user`	Name of the current user, typically the job creator. When a job is created, all its associated tasks are also created. Therefore, any task or workflow attributes that were specified as `iw_user` have their value set to the job creator's user id.

`iw_desc`	Description of the job entered in the job description box when creating a job. This variable is usually used for the workflow's description attribute.
`iw_workarea`	Name of the current workarea (if you create the job from the workarea view, or via submit).
`iw_areaowner`	Name of the user who owns the workarea where a job is crated.
`iw_branch`	Name of the current branch.

Custom Variables

WorkflowBuilder 5.5 includes a Perl Code Editor that enables you to add Perl code (including custom variables) to workflow templates. This functionality is commonly used to control the appearance of the forms associated with your template. Chapter 5, "WorkflowBuilder Tutorial," describes the creation and use of a number of custom variables.

Nested Workflow

TeamSite 5.5.2 enables workflow developers to create *nested workflow*—workflow that is contained within other jobs or tasks. The implementation of nested workflow is similar to external and CGI tasks where the activation of workflow tasks is either automatically or manually instantiated causing an association of a new job with the workflow task. The nesting process creates a parent/child relationship with the task as the parent and the job as the child.

The relationship between a workflow task and its nested workflow includes:

- the ability to pass variables and file lists from the parent task to the child job
- the ability for nested jobs to pass some or all variables and file lists to the parent job upon the child job's completion
- the ability for the child job to cause a transition to occur in the parent task upon the child job's completion
- the lifetime of a nested job is dependent upon its parent task's workflow lifetime—it should not be removed from the backing store until its parent task is deleted

Workflow tasks can either be specified with a path to a *job specification file* or to a *workflow template file* (.wft). In the case of a job specification file, upon activation of the workflow task, TeamSite compiles and instantiates a new job using the specification file. In the case of a workflow template file, the task owner must manually start the task using the New Job window to input job variables and subsequently initiate the job.

Creating Jobs with Nested Workflow

Complete the following procedure to create a job with a nested workflow task. This procedure assumes that the `author_assignment_with_nested_job.wft` file installed with WorkflowBuilderis specified in your `available_templates.cfg` file. If it is not, you can locate the sample workflow file in `iw-home/local/config/wft/examples` and add it to your `available_templates.cfg` file.

The `author_assignment_with_nested_job.wft` file defines a job that contains two tasks, the second of which does not begin until the first has been approved by an editor.

1. Log into TeamSite using the Editor, Master, or Administrator role.
2. Select **File** > **New Job** from the drop-down menu.

   The New Job window is displayed.
3. Complete the following steps in the New Job window:

   a. Select the **Author Assignment with Optional Nested Job** template.

   b. Type a description of the new job, for example: **`Test of Nested Workflow`**.

   c. Click **New Job**.

      This executes the `iwwft_instantiator.cgi` to instantiate the job. The New Job Template window is displayed with the description you entered in step b in the **Job Description** field.
4. Complete the following steps in the New Job Template window:

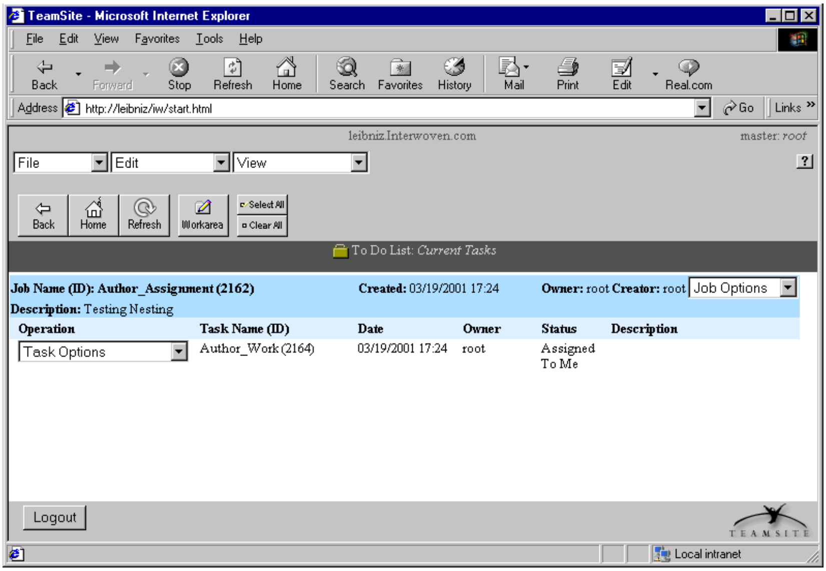
   a. Select an Author from the drop-down menu (to make this demonstration easier, select the same user as you are currently logged in as).

   b. Select a Branch from the drop-down menu.

c. Type the name of a workarea in the Enter Workarea field.

d. Click **Run Job**.

5. In the main TeamSite window, click **To Do** to display the assignment of the job you just created.



Note the following in the graphic:

– The Owner and Creator are both root.

– This screen does not make any mention of the nesting—it is invisible to the person to which the task is assigned.

Chapter 2

Installing WorkflowBuilder

In addition to installing the TeamSite server and client as described in the *TeamSite Administration Guide*, you must install two WorkflowBuilder components to complete the installation. These components are:

- WorkflowBuilder Server—Must be installed on your Solaris, Windows NT, or 2000 server running TeamSite 4.5 or later.
- WorkflowBuilder client application—Must be installed on a Windows 98, NT, or 2000 system that is, or can be, networked to your TeamSite server.

Installation Prerequisites

Ensure these prerequisites are met before installing or upgrading the two WorkflowBuilder components:

- A TeamSite server (version 4.5 or greater) is properly installed and licensed as described in the *TeamSite Administration Guide.*
- You have the WorkflowBuilder Server component that matches your server platform (Solaris, Windows NT, or Windows 2000.
- For the server where TeamSite is installed, you must have Administrator privileges (Windows) or root access (Solaris).
- If you have made changes to a previously installed version of the `available_templates.ipl` file and want to preserve the changes, make a backup copy and rename it. This prevents the WorkflowBuilder Server installation program from overwriting your existing `available_templates.ipl` file.

  You can merge your customizations into the new `available_templates.cfg` file after you have finished installing WorkflowBuilder.

  Note: Do not edit the new version of `available_templates.ipl`.

Installing the WorkflowBuilder Server

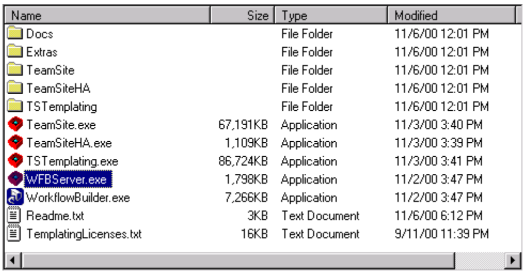
Complete the following procedure that corresponds with your server platform.

Windows NT or Windows 2000 Servers

1. Log on to the system where your TeamSite server is installed as a user with Administrator privileges.

2. Insert the TeamSite CD-ROM.

   The WorkflowBuilder executables are on the same CD-ROM as the TeamSite server.



Name	Size	Type	Modified
Docs		File Folder	11/6/00 12:01 PM
Extras		File Folder	11/6/00 12:01 PM
TeamSite		File Folder	11/6/00 12:01 PM
TeamSiteHA		File Folder	11/6/00 12:01 PM
TSTemplating		File Folder	11/6/00 12:01 PM
TeamSite.exe	67,191KB	Application	11/3/00 3:40 PM
TeamSiteHA.exe	1,109KB	Application	11/3/00 3:39 PM
TSTemplating.exe	86,724KB	Application	11/3/00 3:41 PM
WFBServer.exe	1,798KB	Application	11/2/00 3:47 PM
WorkflowBuilder.exe	7,266KB	Application	11/2/00 3:47 PM
Readme.txt	3KB	Text Document	11/6/00 6:12 PM
TemplatingLicenses.txt	16KB	Text Document	9/11/00 11:39 PM

3. Double-click `WFBServer.exe`.

   The remainder of the installation program is completed automatically.

Solaris Servers

1. Log on to the system where your TeamSite server is installed as the root user.

2. Insert the TeamSite CD-ROM.

   The WorkflowBuilder installation files are on the same CD-ROM as the TeamSite server. The top level of the CD contains the WorkflowBuilder Server installation file.

3. Copy the installation file to the TeamSite installation directory, `iw-home`:

   ```
   % cp wfbserver.5.5.0.BuildNumber.tar.gz iw-home
   ```

4. Decompress the installation file:

   ```
   % gunzip wfbserver.5.5.0.BuildNumber.tar.gz
   % tar xvf wfbserver.5.5.0.BuildNumber.tar
   ```

5. Start the installation program:

   `% ./wfbinstall.sh`

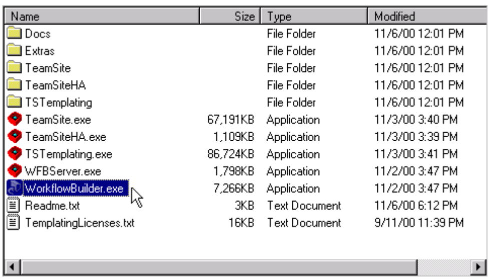
6. Respond to the prompts displayed by the installation program.

Installing the WorkflowBuilder Client

1. Log on to the system where you want to install WorkflowBuilder as a user with Administrator privileges.

2. Insert the TeamSite CD-ROM in your local drive.

   The WorkflowBuilder executables are on the same CD-ROM as the TeamSite server.



3. Double-click `WorkflowBuilder.exe`.

   The WorkflowBuilder installation program prompts you to accept the default installation directory `C:\Program Files\Interwoven\WorkflowBuilder`.

4. Click **Next** to accept the default location, or **Browse** to specify a different location.

   The remainder of the installation program is completed automatically.

Uninstalling WorkflowBuilder

If you are upgrading the WorkflowBuilder client (from version 4.5), you must uninstall the old version before beginning the installation program.

1. Select **Start** > **Settings** > **Control Panel**.
2. Double-click the **Add/Remove Programs** icon.
3. Select **Interwoven Workflow Builder**, and click **Add/Remove**.
4. Click the **Remove** option button.
5. Click **Next**, then **OK** to confirm that you want to remove the WorkflowBuilder client.

Note: You cannot uninstall the WorkflowBuilder server. Whenever you install a new version, it automatically overwrites the existing version.

Chapter 3

WorkflowBuilder GUI

This chapter describes the various GUI elements contained in WorkflowBuilder. The actual procedures for using these features are contained in Chapter 4, “Using WorkflowBuilder,” and Chapter 5, “WorkflowBuilder Tutorial.”

Toolbars

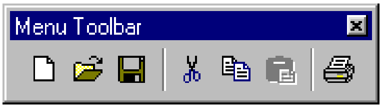
WorkflowBuilder contains the following four toolbars:

- Menu
- Task
- Alignment
- Zoom

To display a toolbar, select it from the **View** menu. You can drag a displayed toolbar anywhere on your desktop.

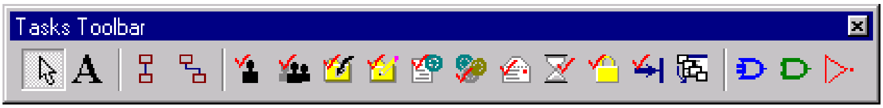
Note: When you move a toolbar outside the WorkflowBuilder GUI, it becomes a floating palette and displays a title bar with a close button (**X**). Clicking the close button hides the toolbar; it does not return it to its default location in the WorkflowBuilder GUI. To return the toolbar into view, select **Tools > Customize**. Select the **Toolbar** tab and check the box next to the corresponding toolbar.

The Menu Toolbar



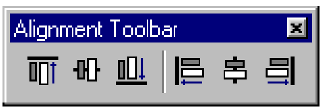
The Menu toolbar contains shortcuts for basic file operations (new, open, and save), edit operations (cut, copy, and save), and print.

The Tasks Toolbar



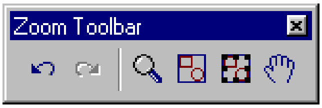
The Tasks toolbar enables you to change your cursor to the selection pointer, place text labels, transition lines, tasks, and conditions on your canvas.

The Alignment Toolbar



The Alignment toolbar is activated when you select two or more objects. Objects are aligned relative to the *last* object selected. The options are top-, center-, and bottom-aligned on the horizontal axis, and left-, center-, and right-aligned on the vertical axis.

The Zoom Toolbar



The Zoom toolbar contains buttons to undo an action or redo an undone action. It also contains three zoom buttons:

- Zoom
  - Click **Zoom** (magnifying glass icon).
  - Move the magnifying glass cursor over the object you want to view.
  - Left click to zoom in (increase magnification), right click to zoom out (decrease magnification).
- Zoom to Fit
  - Click **Zoom to Fit** to resize your view so that all objects are magnified to the maximum possible size within the boundary.
- Zoom to Fit Selected Objects
  - Select the objects on the canvas you want to fit in your current view.
  - Click **Zoom to Selection**. The view is resized so that the selected objects appear magnified to the maximum possible size that keeps them within the boundary.

The Zoom toolbar also contains a Move button (hand icon) that enables you to grip the page and move it up or down.

The View Menu

Features in the **View** menu enable you to control your WorkflowBuilder environment. Use them to open or close windows, hide toolbar items, set grid properties, zoom, and show page boundaries.

Workbook

If you have more than one workflow file open, you can use the Workbook view to display a tab for each open workflow file. The tabs are displayed at the bottom of each file window. Click the tab to display the corresponding file.

When **Workbook** is not checked in the **View** menu, display the file by clicking in that file's window. You can move a file in either view by grabbing its title bar.

Sticky Mode

The Sticky Mode controls whether or not you can place multiple workflow elements (tasks, transitions, conditions) on the canvas without re-selecting the element from the **Tasks** toolbar. This option toggles between on and off positions.

- When this option is checked, each mouse-click adds another of the same element to the canvas until you right-click.
- When this option is not checked, a mouse-click adds one element to the canvas. If you want to add another element, you must re-selecting the element from the **Tasks** toolbar.

Toolbars

Unchecking any of the toolbar items in the **View** menu removes the toolbar from view. You can also move the toolbars around in the WorkflowBuilder window, or display them as floating palettes. Grab the grip on the left hand border of the toolbar to move it.

Status Bar

The status bar is located on the bottom of the WorkflowBuilder window. Uncheck **Status Bar** to hide it from view.

Set Canvas Size

Canvas size is measured by page (8x11). You can set the height and wide of your workflow file to between one and 1000 pages. If you plan to print your workflow file, you may want to limit it to a size and layout suitable for printing.

Grid

You can create a contrasting background to help you align objects on the page. Grid properties such as color, vertical and horizontal spacing, and snapping can be customized in the Grid Properties dialog box.

Snap to Grid

When **Snap to Grid** is checked, the objects you place (or move) on the canvas align with the upper left corner of the nearest point on the grid (points are defined by the intersection of horizontal and vertical lines).

When **Snap to Grid** is not checked, you can place (or move) objects anywhere on the canvas.

Grid Properties

Use the Grid Properties dialog box to customize the grid. You can set visibility, snap options, color, as well as the height and width of grid lines.

Zoom Normal

Select **Zoom Normal** to return the window to the default (100%) view.

Zoom to Fit

Select **Zoom to Fit** to resize the view to a percentage that allows you to view all selected objects without scrolling. First select the objects that you want to fit in view, then select **View** > **Zoom to Fit**.

Zoom Percent

You can resize the view by selecting **View** > **Zoom Percent** and selecting 50%, 75%, 100%, or 200%.

Zoom Custom

You can set a custom view size by selecting **View** > **Zoom Custom** and entering the desired magnification.

Attributes Window

You can open the Attributes window by selecting **View** > **Attributes Window**. The Attributes window is where you specify attributes and values for each workflow element (for detailed information about attributes see "Setting Attributes" on page 60).

Output Window

You can open the Output window by selecting **View > Output Window**. The Output window displays any error messages that are generated when WorkflowBuilder tries to validate your job or workflow template. Typically, these error messages have to do with required attributes of tasks. After saving a job or workflow template, view the Output window to ensure that no errors were generated. If an error message is displayed in the Output window, set the required attributes and save your job or workflow template again.

Perl Code Editor

You can open the Perl Code Editor by selecting **View > Perl Code Editor**. The Editor enables you to enter custom Perl code and to create custom variables. The Perl Code Editor is described in detail in "Defining Custom Variables" on page 88

Where To Go from Here

Having familiarized yourself with the WorkflowBuilder GUI features described in this chapter, you can proceed to either of the following chapters:

- Chapter 4, "Using WorkflowBuilder" contains reference material and procedures for viewing and modifying sample workflow templates, creating your own custom workflow templates, setting attributes and variables, and transferring finished templates to your TeamSite server.
- Chapter 5, "WorkflowBuilder Tutorial" contains a step-by-step tutorial for building a new workflow template based. Many of the features presented in the tutorial are easier to understand within the context of creating a complete workflow template, but it does *not* describe every feature included in WorkflowBuilder. It only describes the features that are specific to the template being created.

Chapter 4

Using WorkflowBuilder

Each workflow template describes a business process that can include user tasks, group tasks, and a wide variety of automated tasks. Using WorkflowBuilder, the actual creation of workflow templates is simple—the more difficult part is negotiating the business rules with the decision makers in your organization, and translating the logic into elements that can be represented within WorkflowBuilder.

This chapter describes the functionality included in WorkflowBuilder including a number of server management features. Additionally, Chapter 5, "WorkflowBuilder Tutorial" discusses many of the same features, but describes them within the context of constructing a single workflow project. Some features are more easily understood within the context presented by the tutorial, but note that features not relevant to the scenario described in the tutorial are not discussed.

Sample Workflow Templates

A good way to plan a custom workflow template is to view the sample templates shipped with WorkflowBuilder. In addition to providing a guide for creating custom workflow templates, these templates can be modified, saved, and sent to your TeamSite server. There are three sample templates that include both the `.wft` and corresponding `.wfb` files. Both of these files are required to display a template in WorkflowBuilder. These three files are installed by the WorkflowBuilder installation program and are located by default in `C:\Program Files\Interwoven\WorkflowBuilder\examples`.

Viewing and Modifying Example Templates in WorkflowBuilder

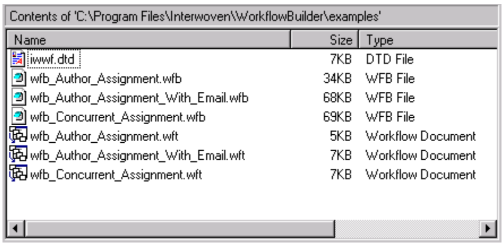
You can open any existing workflow template file (`.wft`) that has a corresponding image file (`.wfb`) and display it in WorkflowBuilder. Complete the following procedure to view a workflow template file.

1. Select **File > Open Workflow**.

   The Open window is displayed.

2. Navigate to the `examples` folder (`C:\Program Files\Interwoven\WorkflowBuilder\examples`).

   The three sample `.wft` files and corresponding `.wfb` files installed with WorkflowBuilder are displayed.

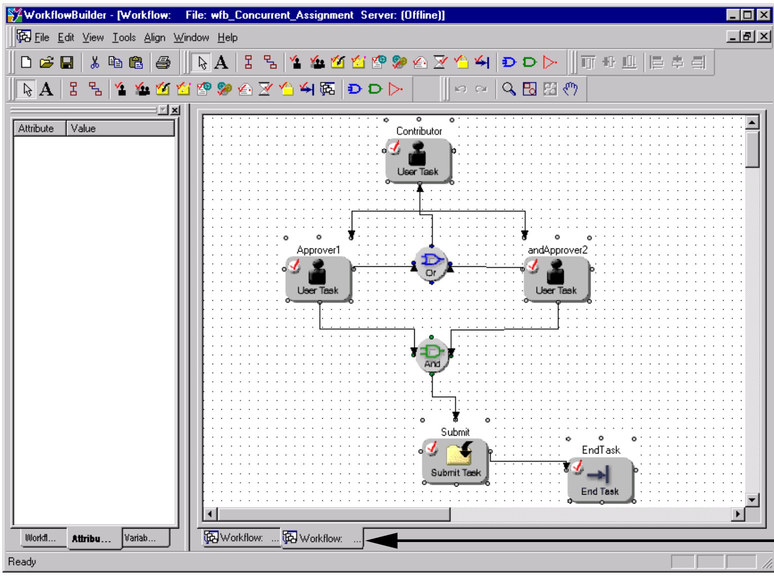
   

3. Select one of the `.wft` files and click **Open**.

   The Login dialog box is displayed.

4. In the Login dialog box, enter your login information, or choose **Offline Mode** (see “Logging In” on page 53).

The selected file is displayed.



Multiple File Tabs

Note: If you open more than one file, each file is stored on a tab view that can be selected at the bottom left of the canvas (as shown in the preceding figure).

5. Optionally, modify any of the elements, transitions, attributes, or variables; then select **File** > **Save As** and create a new template.

Creating New Jobs and Workflow Templates

When you create a workflow template you add *elements* such as tasks and transitions. Each element has a series of *attributes* which must be defined. Some of these attributes are mandatory and some are optional. Transitions between tasks specify when and how the next task in the flow is signaled.

After completing and saving workflow templates, you can send them to your TeamSite server. For information about transferring your workflow templates to TeamSite, refer to "Sending Workflow Templates to the Server" on page 64.

Your workflow template can describe a general workflow model, or it can describe a specific job. The difference between these two is in the way attributes are set.

To create a new job or workflow template:

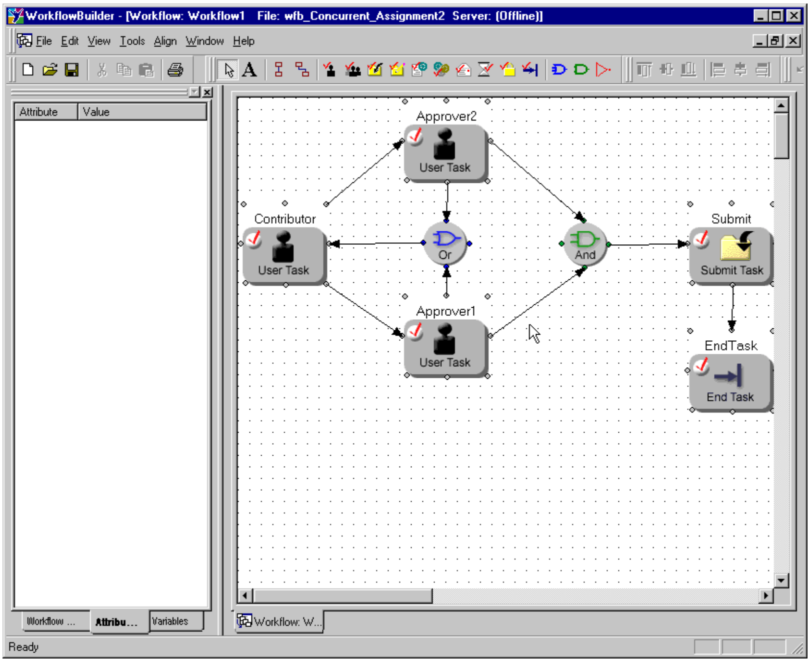
1. Select **File > New Workflow**.
2. In the Login dialog box, enter your login information, or choose **Offline Mode** (see "Logging In" on page 53).
3. Edit the workflow template:

   Select **View > Attributes Window** to display the Attributes window so you can set attributes on the elements you place in your workflow template. If attributes are set with user variables (see "Setting Variables" on page 61), the file will be a workflow template which may be invoked through the **New Job** menu item or the **Submit** button in the TeamSite GUI. If attributes are not set with user variables, the file will describe a specific job.

   (Optional) Select **View > Output Window** to open the Output window to view validation comments as you work.

4. Select elements from the toolbar and place them on the canvas in the order you want (see "Placing Tasks on the Canvas" on page 56 and "Drawing Transitions" on page 56).
5. Set attribute values for each element (see "Setting Attributes" on page 60).
6. When you are done, select **File > Save**. and check the Output window to make sure there are no validation errors. You can then send your workflow template or job specification file to the TeamSite server.

When you save a new workflow template, two files are created: a `.wft` file, and a `.wfb` file which contains the workflow diagram graphic.



A completed workflow template

Logging In

Each time you create or open a workflow template, WorkflowBuilder asks whether you want to log in or work offline. Logging in to WorkflowBuilder allows you to receive information from the TeamSite server to use in your job or workflow, such as lists of users, TeamSite areas where you can create tasks, and lists of available files. If you have a network connection to the TeamSite server, you should log in. If you do not have a network connection, you can work in Offline Mode.

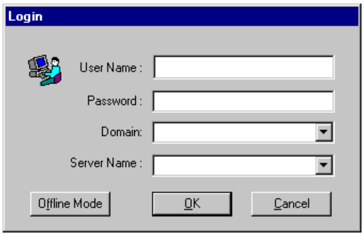
Offline mode enables you to use WorkflowBuilder even when you do not have access to a TeamSite server. However, to use the workflow templates you generate, you will eventually need to be able to connect to the TeamSite server. When you work in offline mode, you do not have access to TeamSite-specific information, such as the lists of users or TeamSite areas.

You can generate workflow templates that do not include this information when you work in offline mode, then update them at a time when you have a connection to the TeamSite server.

To log in (online mode):

1. Select **File** > **Select Workflow** or **File** > **Open Workflow**.

   The Login dialog box is displayed:

   The WorkflowBuilder login dialog

2. Enter your TeamSite username and password.
3. If your TeamSite server is running on Windows NT or Windows 2000, enter your domain in the **Domain** field.
4. Enter the name of the TeamSite server, or choose it from the pull-down menu if you have connected to it before.
5. Enter the port number where you connect to your web server.

   Port 80 and Port 81 are valid entries.
6. Click **OK**.

   The name of the TeamSite server and the port number are displayed in the title bar.

To work in offline mode instead, click **Offline Mode**.

Editing Existing Workflow Templates

You can edit workflow templates that have been created using WorkflowBuilder. You cannot use WorkflowBuilder to edit workflow templates that have not been created using WorkflowBuilder because these files do not contain a workflow diagram.

To open a workflow template:

1. Select **File > Open Workflow**.
2. Navigate to the workflow template you want to edit, and click **Open**.
3. In the Login dialog box, enter your login information, or choose **Offline Mode**.

   The file will open in WorkflowBuilder.
4. Edit the workflow template:

   Select **View > Attributes Window** to display the Attributes window so you can set attributes on the elements you place in your workflow template.

   (Optional) Select **View > Output Window** to open the Output window to view validation comments as you work.
5. Select elements from the toolbar and place them on the canvas in the order you want.
6. Set attribute values for each element.
7. When you are done, select **File > Save**. and check the Output window to make sure there are no validation errors. You can then send your workflow template or job to the TeamSite server.

Placing Tasks on the Canvas

To place an object on the canvas:

1. Select an object from the Tasks Toolbar.

   When you move the pointer to the canvas, a graphic icon is displayed under it to indicate that a mouse click will place an object on the canvas.

2. Click on the area of the canvas where you want to place the object.

   Note: If Sticky Mode is on (**View** > **Sticky Mode)** each click adds another of the same object to the canvas until you right-click.

Drawing Transitions

Transitions indicate the flow from one task to the next. There are four types of transitions: Successor, Timeout, Reset, and Inactivate (as described on page 20). You can qualify transitions by using conditional activation elements such as AND, OR, or NOT.

When drawing transitions, use the Straight Transition button to draw straight lines and the Segmented Transition button to draw a transition around other objects in your diagram.

To draw a transition:

1. Using the Transition buttons in the Tasks Toolbar, select the type of line you want to draw (either straight or segmented). When you move the mouse arrow onto the canvas, it changes to a plus sign ().

2. Click a connection point of the task you want to transition from. When you see the cursor change to crosshairs (), you can click to anchor your transition line.

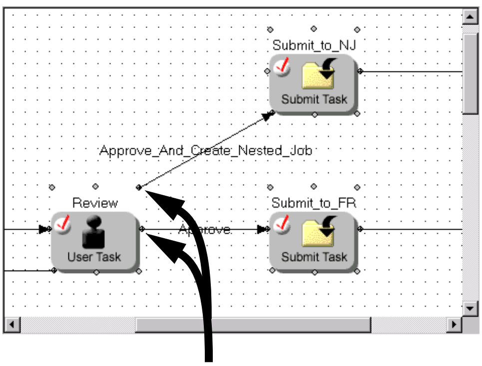
3. Click a connection point of the task you want to transition to.

   Note: A segmented transition provides two joints on the line between the anchor points. Use these joints to position the line around other objects on the canvas.

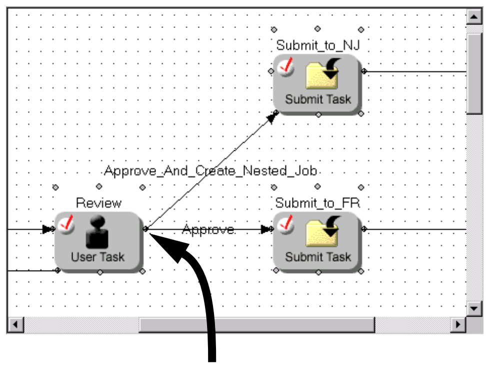
4. When you are done drawing your transition, right-click to reset your mouse to the pointer.

When any task (other than the end task) is completed, the task owner is presented with options for triggering one or more *successor sets*. In WorkflowBuilder, the ports on the task icon represent successor sets. The maximum number of successor sets that a task can have when created using WorkflowBuilder is eight (there are eight ports per icon). Each successor set can have one or more tasks in them. Therefore, when a task is completed, the user is provided with an option of selecting a successor set. When a selection is made, all the tasks in that successor set are started.

In the following graphics, note that the transaction between the Review User Task and its successor tasks (Submit to NJ and Submit to FR) originate from different points.



Submit (successor) tasks originate from different ports on the "Review" User Task. Result: the user chooses which one of the two successor tasks gets started.



Submit (successor) tasks originate from the same port on the "Review" User Task. Result: both of the two successor tasks get started.

- In the graphic on the left (where tasks originate from different ports on the "Review" User Task), when the "Review" User Task is completed, the task owner is given the choice to select which of the two Submit successor tasks must be completed. The selected task is started, and the task not selected does not get started.
- In the graphic on the right (where tasks originate from the same point on the "Review" User Task), when the "Review" User Task is completed, the task owner is given only one choice. It differs from the other example in that Submit successor task is actually a set of two tasks that must both be completed. Therefore, when the task owner selects the only option, both tasks are triggered.

In the workflow segment shown in the graphic, the User Task calls for a review of some work done by an author. The reviewer (typically an Editor) has the choice of approving the work and creating a nested job (Submit to NJ) or approving the work and submitting it for final review (Submit to FR). (Actually there is a third option: the "Review" User Task has a Reject arrow originating from the lower left port of the task icon but is cropped out of the image.)

Adding Text Labels

You can add text labels to objects on the canvas. Text labels, unlike name and description attributes, allow you to enter an unlimited number of characters. You can use labels to add a descriptive name or explanatory notes.

To add text labels:

1. Click the text label button in the Tasks Toolbar.
2. Click on the canvas where you want to place the text label.

   A text area appears with selection boxes around it where you click.
3. To edit the text of the label, move your cursor into the boundaries of the text label and double-click.
4. Right click once when you are finished placing text labels on the canvas.

Moving Objects

You can place objects anywhere on the canvas. You can also use the alignment buttons in the toolbar to align objects on the canvas.

To move an object on the canvas:

1. Move the mouse pointer over the object you want to move. A multidirectional arrow is displayed at the tip of your pointer.
2. Click and drag the object to where you want to place it.

Selecting Multiple Objects

To align objects on the canvas, you must select more than one object. You can also select multiple objects and drag them around the canvas.

To select multiple objects:

- Hold down the Shift key on your keyboard and click on the objects you want to select. When an object is selected, selection boxes appear around the object.

A selected task

Aligning Objects

You can align objects on the canvas by using the alignment buttons in the toolbar.

Option	Description
Align Top	Aligns the top of objects with the top of the last object selected.
Align Horizontal Centers	Aligns the horizontal center of objects with the center of the last object selected.
Align Bottom	Aligns the bottom of objects with the bottom of the last object selected.
Align Left	Aligns objects with the left side of the last object selected.
Align Vertical Centers	Aligns the vertical center of objects with the center of the last object selected.
Align Right	Aligns objects with the left side of the last object selected.

To align objects:

1. Select two or more objects that you want to align. The alignment buttons in the toolbar become active.

   Note: All objects selected will be aligned relative to the object you select last.

2. Use the alignment buttons in the toolbar to align the objects.

Setting Attributes

To set attributes of a task or transition:

1. Select the arrow  in the toolbar.

2. Click the task or transition whose attributes you want to select.

3. Click the **Attributes** tab in the left-hand pane. The attributes for that task or transition are displayed in the Attributes window.

4. Click the **Value** column of the attribute you want to set. Some attributes can be set using a pull-down menu (for example, AreaVPath). If you are connected to the TeamSite server you can set some attributes using the **...** button (for example, AreaVPath or Owner). Other attributes must be typed (for example, Description).

   If you want to use a variable for the value of this attribute, select its name from the pull-down menu.

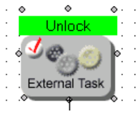
   If you want to use information from the TeamSite server, click the **...** button and select the value you want from the list that appears.

   If you want to type the value in directly, double-click on the Value column of the attribute you want to set, and enter the value. Note that some attributes cannot be set this way (for example, AreaVPath).

All workflow templates must contain at least one Task with the Start attribute set to Yes. Note that a template can contain multiple Start attributes. For example, consider a job that, upon being instantiated, needs to send different pieces of email to four different people and run a process that generates one of the files to be manipulated during the workflow process.

Since none of these tasks have a dependence on the others you can run them in parallel, each with its own Start attribute.

Tasks with Start attributes are displayed using a green text box across the top of the task icon, as shown in the following graphic:



Also note that each template must contain an End task.

Setting Variables

WorkflowBuilder allows you to set three types of variables: system, custom, and user. You can use these variables to specify values of attributes.

Creating System Variables

To set a system variable:

1. Click the **Variable** tab in the Attributes window.
2. Double-click an entry in the **Name** column. Enter the name of your variable.
3. Double-click the corresponding entry in the **Value** column. Select **System Variable** from the pull-down menu.
4. The Select System Variable window will appear. Select the system variable you want to use. Click **OK**.

When the workflow template is transferred to the TeamSite server, the variable will be set to the value you have selected (for example, if you named the variable `userrole` and selected `iw-role` as the system variable, `userrole` would be set to the role of the user who instantiates the job).

Creating Custom Variables

WorkflowBuilder 5.0 includes a Perl Code Editor that enables you to add Perl code (including custom variables) to workflow templates. This functionality is commonly used to control the appearance of the forms associated with your template.

"Defining Custom Variables" on page 88 describes the creation a number of custom variables within the context of the creation of an entire workflow template.

To create custom variables:

1. In the Variables window, click the **Variables** tab.
2. Double-click an empty cell in the **Name** column and enter a name for the new variable.

   Note: Custom variable names should begin with a lower-case "c", for example `cCustom`. Following this convention ensures that the variables you create do not conflict with others included with future releases of WorkflowBuilder.

3. Double-click in the **Value** column and select **Custom** from the pull-down menu.
4. Select **View** > **Perl Code Editor** to open the Perl Code Editor.
5. In the Perl Code Editor, enter the corresponding Perl code to define the variable.

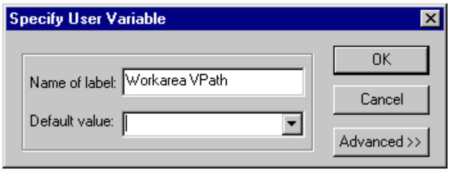
Creating User Variables

To set a user variable:

1. Click the **Variable** tab in the Attributes window.
2. Double-click an entry in the **Name** column.
3. Enter the name of your variable.
4. Double-click the corresponding entry in the **Value** column.
5. Select **User Variable** from the pull-down menu.

   The Specify User Variable dialog is displayed.

6. In the **Name of label** field, enter the text you want to appear next to this variable in the New Job template.



The User Variable dialog

7. Optionally, enter the default value of this variable in the **Default value** field.
8. Optionally, click the **Advanced** button to display the following variable settings:
   - In the **Is this variable required?** section, select **Yes** or **No**.
   - In the **Enter validation rules** section, enter any rules you want to use to determine what types of input are valid. These rules must be specified using Perl regular expressions.
   - In the **Specify error message** section, enter the error message you want to display in the New Job template if the job creator enters an invalid value.
9. Click **OK**.

Configuring Templates to Include Preselected Files

You can configure workflow templates that enable job creators to attach files to jobs when they set job parameters in the TeamSite GUI. These preselected files are automatically attached to each task in the job as the job transitions from task to task.

To configure templates to include preselected files:

1. In the **Attributes** window, click the **Workflow Attributes** tab.
2. In the **Attribute** column, double-click Preselected Files and set the value to **No**.
3. On the canvas, select the task that you want to be the Start task.
4. In the **Attribute** window, click the Attributes tab.

5. In the **Attribute** column, double-click the Start attribute and set the value to **Yes**.

   The **File** attribute of the Start task is automatically set with the Perl variable `$iw_selected_files [];`. WorkflowBuilder prevents you from editing the value of the **File** attribute of any other task in the template after the **Preselected Files** workflow attribute has been set to **Yes** and a Start task has been specified.

Note: Set the value of the **Preselected Files** attribute to **No** if files will be added to the job after it has been instantiated.

Sending Workflow Templates to the Server

After you have created a workflow template, you must transfer its two associated files (the `.wft` and the `.wfb`) to the TeamSite server and decide what constraints (if any) to place on the file to control access to it. When the file is transferred to the server, the `available_templates.cfg` file is automatically updated to reflect that this new (or modified) workflow file is available to specified TeamSite users. The files you send to your TeamSite server are placed in *`iw-home`*`/local/config/wft/wfb`.

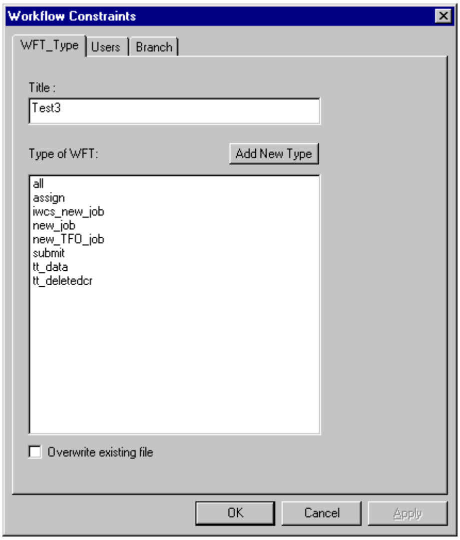
Complete the following procedure to transfer a workflow template to the TeamSite server:

1. Open WorkflowBuilder.
2. Select **File** > **Open Workflow**.
3. Locate and select the `.wft` file you want to send to your TeamSite server (note that recently opened files are available on the **File** menu).

   The Login dialog box is displayed.
4. Log in to your TeamSite server as a Master user (for details about the Login procedure, see page 53).

   The selected file is displayed in WorkflowBuilder.
5. Select **File** > **Send to Server**.

The Workflow Constraints dialog box is displayed with the WFT_Type tab activated.



Workflow Constraints Dialog Box

6. Click OK to accept the default constraint settings (by default, all users may use all workflow templates on all branches) or define a constraint for the selected workflow file as described in "Workflow Template Constraints" on page 66.

7. Choose the type of job your workflow template will create (New Job or Submit Job).

8. Click **OK**.

   The `available_templates.cfg` configuration file is updated on your TeamSite server. The transferred workflow template is available to users connecting to the server using the TeamSite browser-based GUI.

   Note: If you have transferred a job specification file (a file that describes a particular job), the job is instantiated immediately.

For information about creating jobs that use your custom workflow templates (and also the provided templates), refer to the *TeamSite User's Guide* that corresponds with your client platform.

Workflow Template Constraints

TeamSite enables you to control access to specific workflow templates by setting constraints on the workflow template files when they are published to the TeamSite server. A *constraint* is constructed by selecting an entry from each of the following tabs in the Workflow Constraints dialog box:

- Workflow Template Type (WFT_Type)
- Users
- Branch

The Workflow Constraints dialog box is displayed when you send your workflow template files to your TeamSite server (see “Sending Workflow Templates to the Server” on page 64). These constraint types are described in the following sections and the procedure for creating workflow constraints begins on page 68.

Workflow Template Type Constraints

The **WFT_Type** tab of the Workflow Constraints dialog box enables you to determine how each type of workflow template is invoked (for example, through the Submit or New Job commands from the TeamSite GUI). By default, you can control access to the selected workflow template file to following types:

- **all**—Valid for any type of job.
- **assign**—Valid only when you do an Assignment of a file to a user.
- **iwcs_new_job**—Valid when you select the **New Job** option using Interwoven Content Express.
- **new_job**— Valid when you select the **New Job** option from the TeamSite GUI to start a new workflow.
- **new_TFO_job**—Valid when you select the **New Job** option in TeamSite Front-Office.
- **submit**—Valid when a Submit is performed from the TeamSite GUI.
- **tt_data**—Specifies this workflow file be invoked when closing a TeamSite Templating DCR.
- **tt_deletedcr**—Specifies this workflow file be invoked when deleting a TeamSite Templating DCR.

Note: You can add your own custom workflow types by clicking **Add New Type** and entering a new type name in the window that is displayed. The new type is added to the list when you click **OK**.

The **WFT_Type** tab also enables you to assign a title to the workflow template file. The **Title** field defaults to the file name given when the file was created or saved in WorkflowBuilder, but can be changed by editing the name in the Workflow Constraints dialog box. After sending the template to the server, this title is displayed in the list returned by the New Job or Submit Job functionality of the TeamSite browser-based GUI.

For example, if you selected **new_job** as the workflow type, and entered a title of New Custom Workflow, the title is displayed as an option when the end-user selects **File** > **New Job** in the TeamSite GUI.

Template title displayed in New Job window of the TeamSite GUI

The **WFT_Type** tab also contains an Overwrite Existing File option that enables you to overwrite workflow template files already on the server with updated versions of the same file.

User Constraints

The **Users** tab of the Workflow Constraints dialog box enables you to specify which users and types of users (defined by TeamSite roles) can access the workflow template file being sent to your TeamSite server.

The **Users** tab displays the users stored in each of the four TeamSite Roles files (`admin.uid`, `author.uid`, `editors.uid`, and `master.uid`). For more information about defining and managing TeamSite Users, refer to the *TeamSite Administration Guide*.

Branch Constraints

The **Branch** tab of the Workflow Constraints dialog box enables you to specify from which branches the workflow template can be accessed.

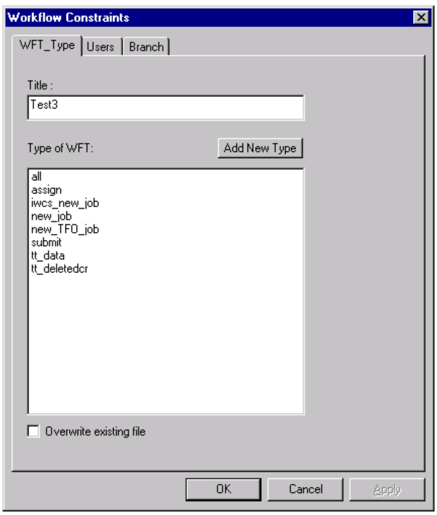
Note: If you do not specify a branch constraint, the template is available from all branches.

Defining a Workflow Constraint

Complete the following procedure to define a workflow constraint for a template file.

1. Open a completed workflow template file from within WorkflowBuilder.
2. Log in to the TeamSite server where you want to send the file.
3. Select **File > Send to Server**.

   The Workflow Constraints dialog box is displayed with the **WFT_Type** tab activated and the file name used as the default title.



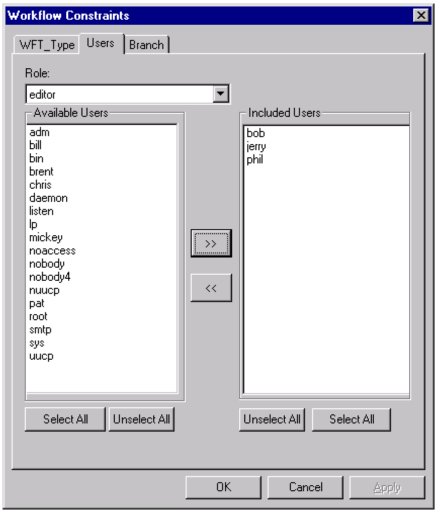
4. Select the type of template you are sending to the server (the eight default types are defined on page 66).

   This selection determines how the workflow template is invoked.

5. Optionally, enter a new title for the template in the **Title** field.

   The entry in the **Title** field is displayed in the TeamSite GUI when the end-user selects the "type" of activity associated with the selection made in the previous step.

6. Click the **Users** tab.



   a. From the **Role** drop-down menu, select the Role of the individual user or group of users to whom you want to allow access to this type (selected in step 4) of workflow file.

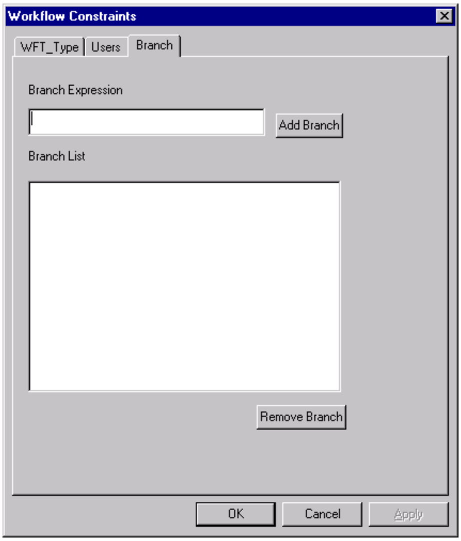
      The **Available Users** list displays all the users with an entry in the Role file that corresponds with your selection. In this example the Editor role is selected.

   b. Click **Select All** or an individual user (you can use Shift+Click to select multiple users) from the **Available Users** list.

   c. Click **>>** to send the selected users to the **Included Users** list.

      In this example, three editors (Bob, Jerry, and Phil) were added to the **Included Users** list.

7. Click the **Branch** tab.



a. In the **Branch Expression** field, type the name of the branch where you want this template to be available, for example, **`main/PressRelease`**.

Note: This branch must already exist on your TeamSite server. This procedure does *not* create the branch.

b. Click **Add Branch**.

The branch you entered is displayed in the **Branch List**.

c. Repeat step a and step b for each branch where you want this template to be available.

d. In the **Branch List**, highlight all the branches where you want this template to be available (you can use Shift+Click to select multiple branches).

e. Click **OK** to send the template to your TeamSite server with the constraints you just defined.

If you need to modify the constraints of a template, open it in WorkflowBuilder (either a local copy or by doing **File** > **Get From Server**) and repeat this procedure this time ensuring the **Overwrite Existing File** option is checked on the **WFT_Type** tab.

Retrieving Files from the Server

You can retrieve copies of workflow template files already transferred to your TeamSite server using WorkflowBuilder. This functionality downloads a copy of both the `.wft` and the `.wfb` files to any client machine with WorkflowBuilder installed. After opening one of these files you can modify it and repost it to the server, or do a **File** > **Save As** to use it as the foundation for creating a new workflow template.

Note: You can only retrieve active templates from the TeamSite server. If the file you want to retrieve is not an active template, you must complete the procedure described on page 72 to change the template's state to active

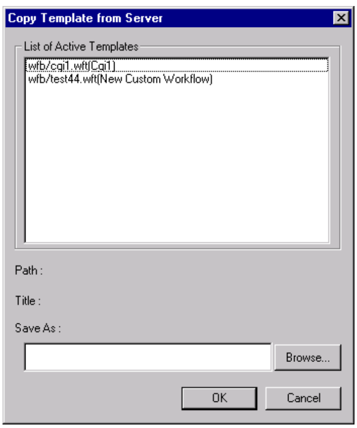
Complete the following procedure to copy workflow templates from your TeamSite server.

1. Select **File** > **New Workflow** to display the Login dialog box.

2. Log in to the server where the workflow template you want to retrieve is stored.

3. Select **File** > **Get From Server**.

   The Copy Template from Server dialog box is displayed.



4. Select the template you want to copy to your client machine.

   The path to the selected file and it's title are displayed next to the corresponding labels. Note the file specifies the `.wft` extension but both the `.wft` and the `.wfb` files are downloaded.

5. Click **Browse** to display the Save As dialog box.

   a. Enter a name for the files in the **File name** field.

      The name selected in this step is used for both template files: one with a `.wft` extension and one with a `.wfb` extension.

   b. Select the location where you want to store the two template files.

   c. Click **Save**.

      The path and the file name you selected are displayed in the **Save As** field of the Open Selected Template dialog box.

6. Click **OK**.

Deleting Files from the Server

You can delete workflow templates from your TeamSite server from within WorkflowBuilder. This functionality deletes both the `.wft` and the `.wfb` files from the server. WorkflowBuilder's Delete Templates dialog box also includes functionality for undeleting files mistakenly deleted and changing the state of a workflow template on your server (valid states are Active and Inactive).

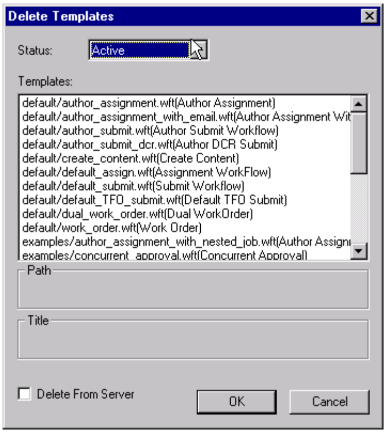
State changes are written to the `available_templates.cfg` file, and are controlled by the include attribute. Active workflow templates have the include attribute set to `yes`. Inactive workflow templates are set to `no`. The following line from the `available_templates.cfg` file shows the state of a file named `LegalApproval.wft`:

```
<template_file active=“yes” name=“LegalApproval” path=“wfb/LegalApproval.wft”>
```

Complete the following “procedure to either delete workflow template files from your TeamSite server or to change their current state.

1. Select **File** > **New Workflow** to display the Login dialog box.
2. Log in to the server where the template files you want to delete are stored.
3. Select **Tools** > **Delete Templates**.

The Delete Templates dialog box is displayed.



4. Delete a workflow template from the TeamSite server:

   a. Set the **Status** field to correspond with the state of the file you want to delete (either Active or Inactive).

      The list of workflow templates displays all the files with the selected status. Note that the list specifies the `.wft` extension but both the `.wft` and the `.wfb` files are deleted.

   b. Select the file you want to delete.

      The path to the selected file and it's title are displayed next to the corresponding labels.

   c. Check the **Delete From Server** box.

   d. Click **OK**.

5. Change the state of a workflow template on the TeamSite server:

   a. Set the **Status** field to correspond with the current state of the file you want to change (either Active or Inactive).

      The **Templates** list displays all the files with the selected status.

   b. Select the file whose status you want to change.

      The path to the selected file and it's title are displayed next to the corresponding labels.

c. Ensure the **Delete From Server** check box is cleared.

d. Click **OK**.

The selected workflow template's entry in `available_templates.cfg` is updated to reflect the change.

Note: Selecting **Tools** > **Undelete Templates** will undo either of these actions: it undeletes the selected file if the **Delete From Server** check box is checked, or reverts to the previous state of the file if the check box is cleared.

WorkflowBuilder Error Codes

The following error codes are displayed by WorkflowBuilder under the specified condition.

Note: Double clicking on a WorkflowBuilder error message displays the task the error occurred in if the error output window is docked (that is, not "floating").

Condition	Error Message
FAILURE	Unknown Reason
WFB_SUCCESS	Success
READ_ERROR	Failed to read the complete file
WRITE_ERROR	Failed to write the compete data to the file
FILE_EXISTS	File already exists
INVALID_FILEPATH	Invalid file path
INSUFFICIENT_PERMISSION	Permission Denied
LOGIN_FAILED	Login failed
INVALID_SESSION	Session invalid
USERDATA_NOT_AVAILABLE	Username not set
HTTP_ERROR	Communication Error
UNKNOWN	Unknown Error
NO_FILE_NAME	Invalid File Name

Condition	Error Message
INVALID_ROLE	Invalid Role
NOT_MASTER	Not Master
OPEN_ROLE_FILE_FAILED	Failed to open role files
CLOSE_SESSION_FAILED	Close session failed
OFF_LINE	Off Line
MEMORY_ALLOCATION_FAILED	Memory allocation Failed
ACCESS_DENIED	Access Denied
DISK_FULL	Disk Full
WFB_EROFS	TeamSite server is frozen
EMPTY_DIRECTORY	Directory is empty
INVALID_ERROR_CODE	Error code is invalid
XML_PARSE_ERROR	XMl Parse Error
SERVER_NOT_INSTALLED	Server Component of WFB is not installed
INVALID_SERVER_ID	Invalid Server ID: TeamSite
FAILURE_GETTING_ARCHIVE_NAME	Failure Getting Archive Name: TeamSite
FAILED_BRANCH_ITERATION	Failed Branch Iteration: TeamSite
NO_BRANCH_NAME	Branch Without A Name!!!: TeamSite
FAILED_WORKAREA_ITERATION	Failed Workarea Iteration: TeamSite
NO_WFB_FILE	The WFB for this template is not available
NO_WORKAREAS	No Workareas

Chapter 5

WorkflowBuilder Tutorial

This tutorial shows you how to create a workflow template and make it available to end-users logged in to your TeamSite server. Use this tutorial to learn the basic skills you will need to develop workflow templates and job specification files, and to learn about some of the features available in WorkflowBuilder 5.5.

For more information about workflow templates and job specification files, see Chapter 7 and Chapter 9.

By the end of this tutorial, you will learn how to perform the following tasks:

- Create variables and specify attributes.
- Use custom variables.
- Control the appearance of the New Job form associated with your template.
- Make your workflow template available to job creators.
- Specify where your templates are invoked and who can invoke them.

The concepts and procedures included in this tutorial are designed to get you through the creation of your first actual workflow template. Some features that are not specific to the creation of this project are not explained. These options—and other advanced WorkflowBuilder features—are described in detail in other chapters of this book.

Prerequisites

This tutorial assumes the following:

- TeamSite 5.5.2 and WorkflowBuilder 5.5 Server are installed and configured as described in the TeamSite Administration Guide, and Chapter 2 of this manual on a system dedicated for WorkflowBuilder training.
- WorkflowBuilder 5.5 client is installed as described in Chapter 2.
- You have Master privileges in TeamSite on the system dedicated for WorkflowBuilder training.
- You have the TeamSite server name and the Interwoven Web daemon (`iwwebd`) port number available.
- You are familiar with basic TeamSite administration tasks (or have access to the TeamSite documentation).

Setting up the Tutorial Environment

1. Add yourself and a fictional user to the Master role file in TeamSite.

   See the TeamSite Administration Guide for instructions about adding users to TeamSite.

2. Establish two branches: **`main/WFB_training1`** and **`main/WFB_training2`**.

   See the TeamSite Administration Guide for instructions about creating branches.

3. In each branch create workareas for yourself and the fictional user.

   See the TeamSite Administration Guide for instructions about creating workareas.

4. In your workarea on the `main/WFB_training1` branch, create a file.

   This file can be any file format and does not need to contain content. It will be used later in the tutorial to show how workflow developers can enable job creators to specify, when a job is created, which files they want attached to all of the tasks in the job.

Tutorial Overview

In a real-world implementation, the development and use of WorkflowBuilder workflow templates follows this pattern:

- **Development**—A workflow developer creates a workflow template that describe the flow of tasks in a particular job.
- **Deployment**—The workflow developer makes the workflow template available to job creators by adding the template to the TeamSite server.
- **Job Instantiation**—In the TeamSite GUI, a job creator selects the template that describes the job to be created. A New Job form is displayed into which the job creator enters data specific to that job.

Although this tutorial only covers the development and deployment phases, the following sections use the tutorial project to describe the three phases of workflow development and use so that you can gain a broad understanding of this process.

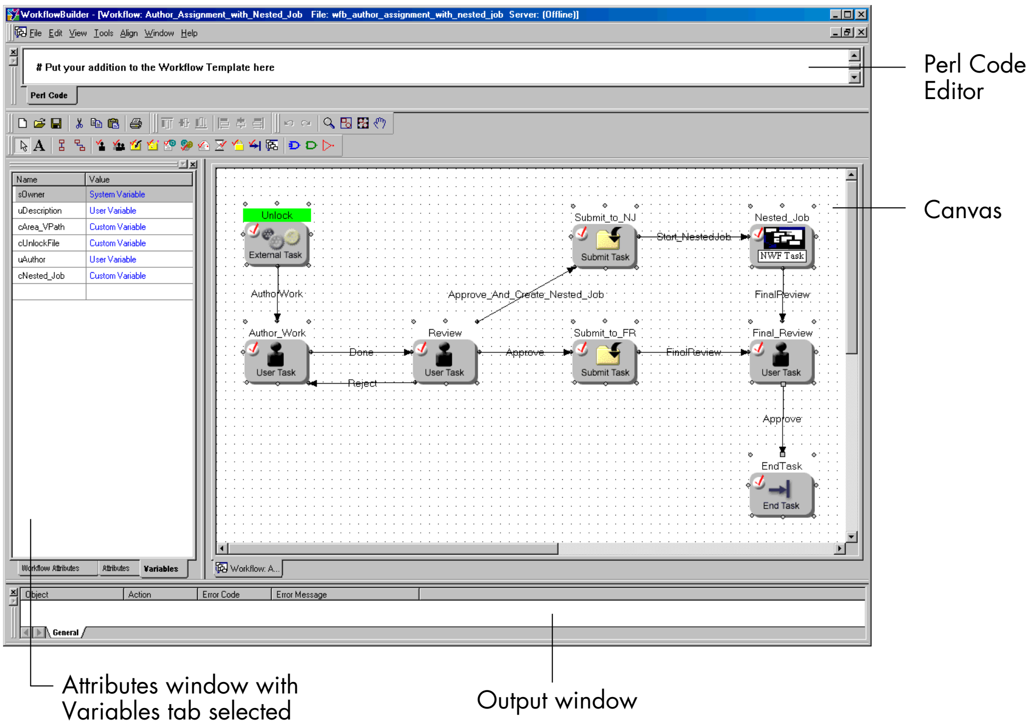
Development

In this tutorial you, the workflow developer, will develop a template that describes a job composed of these tasks:

- **Unlock**—If files have been attached to the job by the job creator, they are unlocked.
- **Author Work**—A task to edit files appears in an Author's To Do List in the TeamSite GUI. The Author completes the requested task and marks it Done.
- **Review**—The job transitions to a reviewer who chooses to either:
  - Reject the Author's changes and return the job back to the Author.
  - Approve the Author's changes.
- **Submit to New Job** or **Submit to Final Review**—Upon approval of the Author's changes, the reviewer chooses to either:
  - Submit the changes and transition the job to a nested workflow. For details about nested workflows, see Appendix B, "Creating a Nested Job."
  - Submit the changes and transition the job directly to a final reviewer.

- **Final Review**—Depending on the action taken by the first reviewer, one of two things will happen:
  - The job described in the nested workflow is started. After that sub-workflow ends, the job transitions to the final reviewer.
  - The job transitions directly to the final reviewer.
- **EndTask**—The job ends upon approval by the final reviewer.

The completed tutorial project should appear as follows:



Completed tutorial project

Deployment

When the Author Assignment with Nested Job template is complete, you will send it to the TeamSite server, at which time you will specify:

- Title of the workflow
- Type of workflow it is
- Which users can access the template
- Branches where the workflow can be run

Instantiation

When job creators initiate a new job, the creator first selects a template in the TeamSite GUI then fills in the required job parameters (such as, job description, task owner, and so on) in the New Job form.

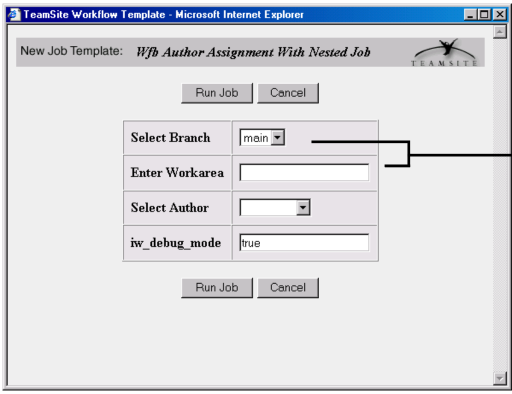
You will configure the tutorial workflow template in such a way that some job parameters will be extracted automatically by the template, and others must be supplied by the job creator.

Additionally, your workflow template will contain a custom variable that changes the appearance of the New Job form under certain conditions.

- If the job creator has attached pre-selected files to the job, the variable automatically extracts the `vpath` to the end user's workarea. In this case, the New Job form does not contain input fields for branch and workarea information.
- If files have not been attached, this information cannot be automatically extracted, so the job creator must specify branch and workarea information in the New Job form (see Figure 2).

You can control such aspects of New Job forms as:

- The conditions under which a form element is displayed.
- The type of form element that is displayed for any given line of input (a text area, for instance, instead of a text field.)
- The label that is displayed for each form element.



These two input fields are conditional. That is, they appear only when a certain condition is met; in this case it is when the job creator has not attached files to the job.

New Job form

Creating a New Workflow

To complete this tutorial, you must work in Online mode. For details about **Online** and **Offline** mode, “Sending Workflow Templates to the Server” on page 64.

To open the tutorial and begin your project:

1. Launch WorkflowBuilder.

   Select **Start** > **Program Files** > **Interwoven** > **WorkflowBuilder**.

   The **Login** dialog box is displayed.

2. Enter the following information in the **Login** dialog box:
   - **User Name**—Your TeamSite user name.
   - **Password**—Your TeamSite password.
   - **Domain**—The domain where the TeamSite sever you are accessing resides. Contact your TeamSite administrator if you do not know the domain where your TeamSite server resides.
   - **Server Name**—The name of the TeamSite server.
   - **Port Number**—The Interwoven Web daemon (`iwwebd`) port number.

3. Click **OK**.

4. Select **File > New Workflow**.

   The attributes window is displayed on the left and the canvas on the right.

5. Place these tasks and transitions on the canvas so that your workflow diagram is identical to the one in figure 3:

   – Task 1—External task

   – Task 2—User task

   – Task 3—User task

   – Task 4—Submit task

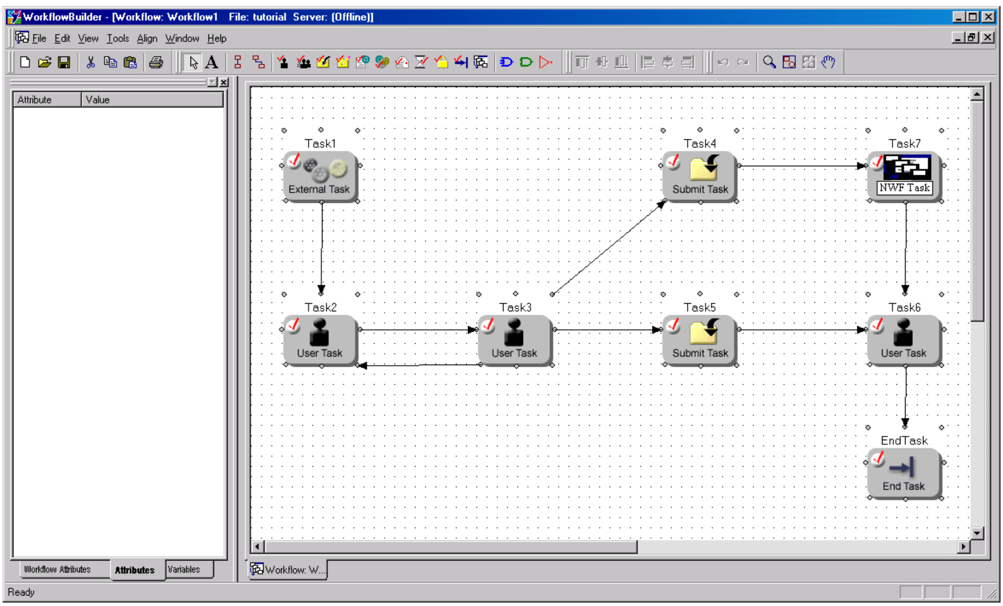
   – Task 5—Submit task

   – Task 6—User task

   – Task 7—NWF (New Workflow) task

   – EndTask

   See "Placing Tasks on the Canvas" on page 56 for information about placing objects on the canvas.



Tutorial workflow diagram

Variables Overview

The variables you create in this section will be used to specify task and workflow attributes in the following sections. Create these variables in the **Variables** tab of the Attributes window (see “Completed tutorial project” on page 80.)

- **sOwner**—System variable. Specifies the owner of the job by automatically extracting the user name of the job creator from the job creator’s system.
- **uDescription**—User variable. Creates an input field in the New Job form where the job creator can enter a description of the job.
- **cArea_VPath**— Custom variable. Extracts the `vpath` information if pre-selected files are attached to the job. Also affects the appearance of the New Job form.
- **cUnlockFile**—Custom variable. If files are attached to the job, this variable unlocks them.
- **uAuthor**—User variable. Specifies the owner of the **Author_Work** task.
- **cNested_Job**—Custom variable. Specifies which `.wft` file to run within the encapsulating job.
- **cTextArea**—Custom variable. Changes the New Job form so that a text area is displayed as the input field for a user variable rather than the default input field.

Naming Conventions

Each variable begins with a lower-case letter that corresponds to the type of variable it is (“s” for System, “u” for User, and “c” for Custom). It is recommended that you use this convention when naming the variables you create. You should also avoid giving variables names identical to attribute names.

Custom Variables

WorkflowBuilder 5.0 enables you to add Perl code to workflow templates so that you can enhance them with custom variables. “Defining Custom Variables” on page 88 includes the Perl code needed to define the custom variables in this tutorial.

For more information, see “Variables” on page 34.

Creating the sOwner Variable

To create the `sOwner` variable:

1. In the Attributes window, click the **Variables** tab.
2. In the **Name** column, double-click in an empty cell.
3. Enter **`sOwner`**.
4. Click in the corresponding row in the **Value** column.
5. Select **System Variable** from the drop-down menu.

   The Specify System Variable dialog box is displayed.
6. In the Specify System Variable dialog box, select **iw_user**.
7. Click **OK**.

Creating the uDescription Variable

To create the `uDescription` variable:

1. In the Attributes window, ensure the **Variables** tab is selected.
2. In the **Name** column, double-click in an empty cell.
3. Enter **uDescription**.
4. Click in the corresponding row in the **Value** column.
5. Select **User Variable** from the drop-down menu.

   The Specify User Variable dialog box is displayed.
6. In the Specify User Variable dialog box, enter **`Description`** in the **Name of label** field.

7. In the **Default value** field, enter **`$cTextArea`**.

   Note: The `cTextArea` variable is used to customize the type of form element that will display for the `uDescription` user variable in the New Job form. It will not be used to specify task or workflow attributes. You will define this variable, along with the other custom variables, in the "Defining Custom Variables" section.

8. Click **OK**.

Creating the cArea_VPath Variable

To create the `cArea_VPath` variable:

1. In the Attributes window, ensure that the **Variables** tab is selected.
2. In the **Name** column, double-click in an empty cell.
3. Enter **`cArea_VPath`**.
4. Click in the corresponding row in the **Value** column.
5. Select **Custom variable** from the drop-down menu.

Creating the cUnlockFile Variable

To create the `cUnlockFile` variable:

1. In the Attributes window, ensure that the **Variables** tab is selected.
2. In the **Name** column, double-click in an empty cell.
3. Enter **`cUnlockFile`**.
4. Click in the corresponding row in the **Value** column.
5. Select **Custom variable** from the drop-down menu.

Creating the uAuthor Variable

To create the `uAuthor` variable:

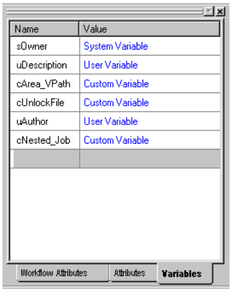
1. In the Attributes window, ensure that the **Variables** tab is selected.
2. In the **Name** column, double-click in an empty cell.
3. Enter **`uAuthor`**.
4. Click in the corresponding row in the **Value** column.
5. Select **User Variable** from the drop-down menu.

   The Specify User Variable dialog box is displayed.
6. In the Specify User Variable dialog box, enter **`Enter user`** in the **Name of label** field.
7. In the **Default value** field, select **$authors** from the drop-down menu.
8. Click **OK**.

Creating the cNested_Job Variable

To create the `cNested_Job` variable:

1. In the Attributes window, ensure that the **Variables** tab is selected.
2. In the **Name** column, double-click in an empty cell.
3. Enter **`cNested_Job`**.
4. Click in the corresponding row in the **Value** column.
5. Select **Custom variable** from the drop-down menu.



Completed Variables tab

In the next section you will define the custom variables you just created.

Defining Custom Variables

WorkflowBuilder 5.0 enables you to you to add custom Perl code. In this section you will add Perl code that defines the custom variables that you created in the previous section:

- `cArea_VPath`
- `cUnlockFile`
- `cNested_Job`
- `cTextArea`

To add the Perl code that defines your custom variables:

1. Select **View > Perl Code Editor**.
2. In the Perl Code Editor, select the following text:

   ```
   # Put your addition to the Workflow Template here.
   ```

3. Enter a definition for each of your custom variables as follows:

<table>
<tr><th>For this variable:</th><th>Enter this:</th></tr>
<tr><td>cArea_VPath</td><td><pre>use TeamSite::Usertask qw(
                          cleanup_paths
                               get_names_from_file
                          get_mail_cmd
                          make_branchpathlist
                          );

sub set_area{
    my($btag, $watag) = @_;
    my($avpath, $bpath, $wapath, $skip);
    my($iwbpath, $iwwapath) =(__VALUE__("iw_branch"),
                              __VALUE__("iw_workarea"));

     if ((length($iwbpath)) > 0 && (length($iwwapath)) > 0){
        $bpath = $iwbpath;
        ($wapath = $iwwapath) =~ s|^\s*/.*:||;
        $wapath =~ s|/\s*$||;
        return("$wapath", "$iwbpath", "$wapath", "TRUE");
    }
    ($bpath, $wapath, $avpath) = cleanup_paths(__VALUE__("$btag"),
                                               __VALUE__("$watag"));

     return("$avpath", "$bpath", "$wapath", "FALSE");
}

my($cArea_VPath, $branch_path, $work_area, $skip_branch) =
set_area("branch_path", "work_area");</pre></td></tr>
<tr><td>cUnlockFile</td><td><pre>my $cUnlockFile = "$iwhome/iw-perl/bin/iwperl $iwhome/local/bin/
unlock.ipl";</pre></td></tr>
<tr><td>cNested_Job</td><td><pre>my $cNested_Job = "$iwhome/local/config/wft/default/
author_assignment.wft";</pre></td></tr>
<tr><td>cTextArea</td><td><pre>my $cTextArea = "<textarea rows='5' cols='40'></textarea>";</pre></td></tr>
</table>

Continue the tutorial by specifying the workflow attributes of this template.

Specifying Workflow Attributes

In this section you will specify four workflow attributes:

- **Name**—Specifies the name that displays in the list of available templates in the TeamSite GUI.
- **DebugMode**—Specifies whether DebugMode is on or off.
- **Description**—Allows the job creator to enter a description of the new job.
- **Owner**—Specifies the owner of the new job as the current user.

These attributes specify the general parameters of the job. For details about workflow attributes, see "Workflow Elements" on page 19.

To specify the workflow attributes:

1. In the Attributes window, click the **Workflow Attributes** tab.
2. In the **Attribute** column, select the attribute you want to specify.
3. Click in the corresponding cell in the **Value** column.
4. Enter the values as described in the following table:

To specify this attribute:	Do this:
`Name`	Enter a name, for example, **`MyTutorialProject`**.
`DebugMode`	Select **Yes** from the drop-down menu.
`Preselected Files`	Select **Yes** from the drop-down menu.
`Variables`	Do not specify this attribute because you will not use Activity variables in this tutorial.
`Description`	Select **$uDescription** from the drop-down menu.
`Owner`	Select **$sOwner** from the drop-down menu.

Continue the tutorial by specifying the attributes of each task on the canvas.

Specifying Task Attributes

In this section you will specify the attributes of each task. Each kind of task has a different set of attributes. For details about task attributes, see “Task Attributes” on page 22.

As you work through this section, it might help you to refer to “Tutorial workflow diagram” on page 83.

To specify the task attributes:

1. In the Attributes window, click the **Attributes** tab.
2. On the canvas, select **Task 1**.
3. Specify the attributes by selecting the one you want in the **Attribute** column, then clicking in the corresponding cell in the **Value** column. Specify each attribute as follows:

To specify this attribute:	Do this:
`Name`	Enter **`Unlock`**.
`Description`	Enter **`Unlock attached files`**.
`Owner`	Select **$sOwner** from the drop-down menu.
`Lock`	Select **No** from the drop-down menu.
`Retry`	Select **Yes** from the drop-down menu.
`Start`	Select **No** from the drop-down menu. **Note:** Set this attribute to **No** so that in a later section you can see how the Validate Template feature works. In that section you will reset the value of this attribute to what is should be—**Yes**.
`AreaVpath`	Select **$cArea_VPath** from the drop-down menu.
`Command`	Select **$cUnlockFile** from the drop-down menu.
`Files`	You cannot set this attribute because the workflow attribute, **PreselectedFiles** is set to **Yes**. It will be set automatically when you complete the section “Saving Your Template” on page 97.

To specify this attribute:	Do this:
`Variables`	Do not specify this attribute because you will not use Activity variables in this tutorial.

4. On the canvas, select **Task 2** and specify these attributes as follows.:

To specify this attribute:	Do this:
`Name`	Enter **`Author_Work`**.
`Description`	Select **$uDescription** from the drop-down menu.
`Owner`	Select **$uAuthor** from the drop-down menu.
`Lock`	Select **Yes** from the drop-down menu.
`Readonly`	Select **No** from the drop-down menu.
`Start`	Select **No** from the drop-down menu.
`AreaVpath`	Select **$cArea_VPath** from the drop-down menu.

5. On the canvas, select the **Task 3** and specify these attributes as follows:

To specify this attribute:	Do this:
`Name`	Enter **`Review`**.
`Description`	Enter **`Review`**.
`Owner`	Select **$sOwner** from the drop-down menu.
`Lock`	Select **Yes** from the drop-down menu.
`Readonly`	Select **Yes** from the drop-down menu.
`Start`	Select **No** from the drop-down menu.
`AreaVpath`	Select **$cArea_VPath** from the drop-down menu.

6. On the canvas, select the **Task 4** and specify these attributes as follows:

To specify this attribute:	Do this:
`Name`	Enter **`Submit_to_NJ`**. Submit_to_New_Job is too long to display as task name, thus the abbreviation "NJ" for "New Job."
`Description`	Enter **`ContentApproved_StartNewJob`**.
`Owner`	Select **$sOwner** from the drop-down menu.
`Start`	Select **No** from the drop-down menu.
`Skip Conflicts`	Select **No** from the drop-down menu.
`Skip Locked`	Select **No** from the drop-down menu.
`Override`	Select **No** from the drop-down menu.
`Unlock`	Select **Yes** from the drop-down menu.
`SaveComments`	Select **Yes** from the drop-down menu.
`AreaVpath`	Select **$cArea_VPath** from the drop-down menu.

7. On the canvas, select the **Task 5** and specify the attributes as you did for the **Submit_to_NJ** task in step 9 on page 93, with the following changes:

To specify this attribute:	Do this:
`Name`	Enter **`Submit_to_FR`**. Submit_to_Final_Review is too long to display as a task name, thus the abbreviation "NJ" for "New Job."
`Description`	Enter **`ContentApproved_StartFinalReview`**.

8. On the canvas, select the **Task 7** and specify these attributes as follows:

To specify this attribute:	Do this:
`Name`	Enter **`Nested_Job`**.
`Description`	Enter **`Nested_Author_Assignment_Workflow`**.
`Owner`	Select **$sOwner** from the drop-down menu.
`Start`	Select **No** from the drop-down menu.
`AreaVPath`	Select **$cArea_VPath** from the drop-down menu.
`Wffile`	To set the `Wffile` attribute: a. Click in the corresponding cell in the **Value** column. b. Click **...**. A dialog box is displayed. c. In the dialog box, select **$cNested_Job** from the drop-down menu. d. Select **WF Template**. e. Click **OK**.

9. On the canvas, select the last **Task 6** and specify these attributes as follows:

To specify this attribute:	Do this:
`Name`	Enter **`Final_Review`**.
`Description`	Enter **`Final_Review`**.
`Owner`	Select **$sOwner** from the drop-down menu.
`Lock`	Select **No** from the drop-down menu.
`ReadOnly`	Select **Yes** from the drop-down menu.
`Start`	Select **No** from the drop-down menu.
`AreaVPath`	Select **$cArea_VPath** from the drop-down menu.

Continue the tutorial by specifying the transitions between the tasks.

Specifying Transitions

There are four types of transitions:

- Successor (default)
- Timeout
- Inactivate
- Resets

Successor transitions are *not* listed with the other types in the drop-down menu because they are applied automatically when no other type is selected. For details about transitions, see“Transitions” on page 20.

All of the transitions in this tutorial are successor transitions. However, you will specify a name for each transition.

To specify a name for a transition:

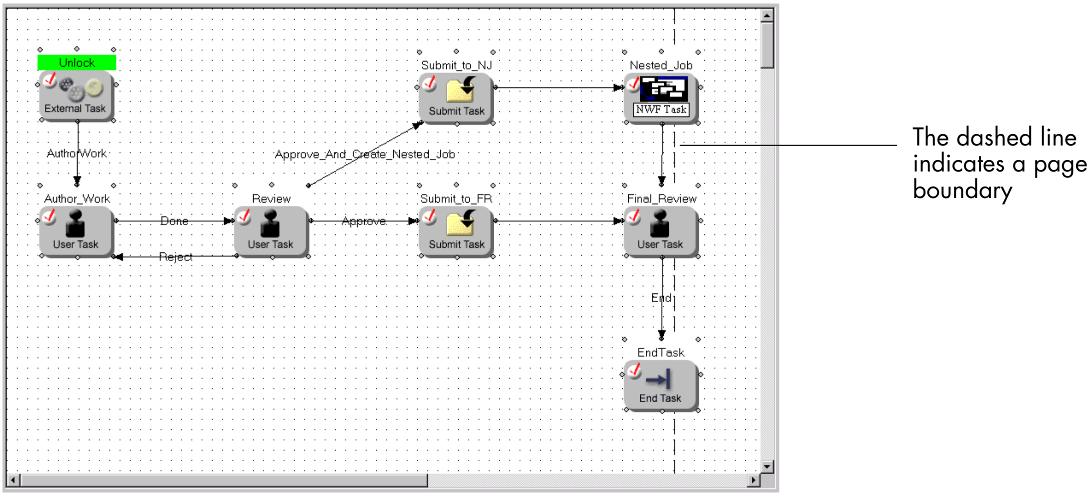
1. On the canvas, select a transition arrow.
2. In the Attributes window, click the **Attributes** tab.
3. In the **Attributes** column, select **Type**.
4. Click in the corresponding cell in the **Value** column.
5. Enter a name for the transition. Refer to “Completed tutorial project” on page 80 for the names for each transition.

Continue the tutorial by printing your template.

Printing Your Template

In this section you will learn how to view page bounds, set the page size, and print your template.

To view page bounds select **View** > **Page**.



Note that the three task icons on the right flow over the boundary of the first page. For this tutorial you'll want to fit everything onto one page.

To fit all the icons onto one page:

1. Hold down the Shift key and select the **Nested_Job**, **Final_Review**, and **EndTask** tasks.
2. Drag them to the left, into the boundary of the first page.

To set the page size of the canvas:

1. Select **View** > **Set Canvas Size**.
2. In the Set Canvas Size dialog box, set the **Horizontal** and the **Vertical** page limits to **1**.
3. Click **OK**.

To print the workflow diagram:

1. Select **File > Print**.
2. In the **Print** dialog box, click **Print**.

Continue the tutorial by saving your work.

Saving Your Template

In this section you will save your workflow file and use the Verify Template feature to find and correct an error in the template.

When you save a workflow file, WorkflowBuilder checks for errors. If errors exist, the Output window opens and displays information about each error (see the graphic on page 80.) You can validate your workflow file at any time by selecting **Tools > Verify Template**.

The error in this tutorial is that no task is identified as a Start task. To fix this, you will reset the value for the `Start` attribute of the **Unlock** task to **Yes**.

1. Save your Template:

   a. Select **File > Save As**.

   b. In the Save As dialog box, navigate to the *`WorkflowBuilder-home`*`\examples` directory.

   c. Name the file **`completed_tutorial.wft`**.

   d. Click **Save**.

   The Output window opens to display an error.

2. Fix the error:

   a. On the canvas, select the **Unlock** task.

   b. In the Attributes window, click the **Attributes** tab.

   c. In the **Attributes** column, double-click **Start**.

   d. In the corresponding cell of the **Value** column, select **Yes** from the drop-down menu.

3. Select **File** > **Save**.

   The Output window lists no errors.

Now that you have verified your template and saved it, you are ready to send it to the TeamSite server.

Sending Your Template to the TeamSite Server

When you are ready to make a template available to TeamSite users, you must transfer its two associated files (the `.wft` and the `.wfb` files) to the TeamSite server and decide what constraints (if any) to place on the file to control access to it. For more information about controlling access using constraints, see "Workflow Template Constraints" on page 66.

In this section you learn how to:

- Specify a title for your template. The title is displayed in the list of available templates in the TeamSite GUI, and can be different than the file name and the **Name** workflow attribute.
- Specify yourself as the only user permitted to access the template.
- Specify that the template can be accessed only from the `main/`*`WorkflowBuilder_Tutorial`* branch.

To send the template to the server, you must be in online mode. If you are not already in online mode, Select **File** > **New** to access the **Login** dialog box.

To send your template to the server:

1. Select **File > Send to Server**.

   The Workflow Constraints dialog box is displayed with the **WFT_Type** tab activated.

2. In the **Title** field, enter **`Tutorial Template`**.

3. In the **Type of WFT** field, select **all**. This makes the template accessible only through the **New Job** option in the TeamSite GUI.

4. Click the **Users** tab.

5. From the **Role** drop-down menu, select **master**.

   All users with an entry in the Master role file are displayed in the **Available Users** field.

6. Select your user name from the list and click >> to add yourself to the list of **Included Users**.

   Do not include the fictional TeamSite user you created when you established the tutorial environment (see “Setting up the Tutorial Environment” on page 78).

7. Click the **Branch** tab.

8. Enter **`main/`*`WFB_training1`***. This restricts access to your the template to this branch.

   Your template will not be accessible from `main/`*`WFB_training2`*.

9. Click **OK** to send your template to the server.

Finish the tutorial by testing the work you’ve done. In the next section you will invoke your template from the TeamSite GUI and create a new job with it.

Testing Your Work

Test your work by creating a new job with your template in the TeamSite GUI.

1. Open a browser and log in to TeamSite.
2. Navigate to your workarea on the `main/WFB_training2` branch.
3. Select the file you created there (see "Setting up the Tutorial Environment" on page 78) to attach it to the job.
4. Navigate to your To Do list.
5. Select **File > New Job**.

   The New Job window is displayed with the Tutorial Template included in the list of available templates.
6. Select **Tutorial Template** and enter a description (such as, ***`test1`***.)
7. Click **New Job**.

   A New Job form is displayed. Notice that you do not need to enter branch or workarea data because you have pre-selected a file that will be attached to the job. Notice also that the form element for **Description** is a text area and not a text field.
8. Enter a description and specify yourself as the task owner.
9. Click **Run Job**.
10. Navigate to the To Do List screen.

    The job you just created has placed a task in your To Do list. Notice that the file you pre-selected is included in the task.

Try to access your template from the other branch, or log out and log back in to TeamSite as the fictional user and try to invoke the Tutorial Template as that user. If you've followed the steps in this tutorial correctly you'll find that your template is inaccessible.

Congratulations! You successfully completed the WorkflowBuilder tutorial.

Chapter 6

Workflow Configuration Files

TeamSite's workflow functionality uses three configuration files to store information about the availability of workflow templates on your TeamSite server. These files are:

- `available_templates.cfg`—XML file installed as part of the WorkflowBuilder installation procedure which stores information about the conditions under which users can access the templates.
- `available_templates.ipl`—PERL file installed as part of the TeamSite installation procedure which parses the `available_templates.cfg` file and returns the names of templates that are valid for the current user.

  Note: Unlike previous versions of TeamSite, the `available_templates.ipl` is *not* user-configurable. All modifications—whether made manually with a text editor, or with WorkflowBuilder—are made to the `available_templates.cfg` file.

- `available_templates.dtd`—File used by `available_templates.cfg` that contains a collection of declarations (elements and attributes) which describe the expected document structure.

Additionally, there are a number of workflow configuration settings that must be made in TeamSite's `iw.cfg` file.

These files are described in detail in the following sections.

The available_templates.cfg File

As described in Chapter 4, WorkflowBuilder contains a number of server-related features that make modifications to the `available_templates.cfg` file. These features enable you to:

- Send and retrieve workflow templates from the server.
- Change the state of a workflow template on the server (valid states are active or inactive).
- Delete files from the server.
- Define constraints that control access to each workflow template based on any combination of user, role, branch, and type (how the workflow is instantiated, for example, by starting a new job).

The `available_templates.cfg` is an XML file which can be modified in a text editor if you prefer. Before making manual modifications to the file, ensure you understand the structure and contents of the file as described in this chapter.

By default, the `available_templates.cfg` file is located in:

- `C:\Program Files\Interwoven\TeamSite\local\config\wft` (Windows servers)
- `iw-home/local/config/wft` (UNIX servers)

available_templates.cfg Structure

The `available_templates.cfg` file is basically a list of workflow templates stored on the TeamSite server. Every workflow template referenced in the `available_templates.cfg` file contains a file name, location, and title. This information is displayed using slightly different syntax depending on whether the template was created and transferred to the server using WorkflowBuilder, or if it is a default workflow template.

- Default workflow templates use the following form:

File Name

```
<template_file name='Author Submit Workflow' path='default/author_submit.wft'>
```

Title File Location

- Files transferred to the server using WorkflowBuilder use the following form:

File Name

```
<template_file active="yes" name="Custom" path="wfb/custom.wft">
```

Status Title File Location

Note: The "default" syntax does *not* specify that the file is active since the default setting (as defined in the `available_template.dtd` file, shown on page 111) is `active="yes"`. If you use WorkflowBuilder to change the status of a workflow template, it will update the file using the WorkflowBuilder-style syntax.

In addition to the file name, location, and title, each workflow template includes the following four lists of rules:

- `command_list`
- `role_list`
- `user_list`
- `branch_list`

Every workflow template listed in the `available_template.cfg` file has a `template_file` element with rules defined in corresponding subelements. Each subelement in the list indicates if the template will be shown when the end-user input matches the element by having a value of "`yes`" ("`yes`" indicating that the template should be listed and "`no`" indicating that the template should not be listed). The template is shown in the list of available templates only if the results of the four lists are "`yes`".

command_list element

The `commmand_list` is similar to the `user_list` element, except that it has no dependencies with `role_list`. The following code specifies that the associated workflow template be active when performing a Submit and that it cannot be invoked by other means:

```
<command_list>
     <command value='submit' />
     <command value='all' include='no' />
</command_list>
```

The valid command values to which you can associate a workflow are:

- `submit` (submitting files)
- `assign` (assign button)
- `new_job` (new job, for WebDeskPro)
- `new_TFO_job` (new job, for TeamSite FrontOffice)
- `tt_data` (saving TeamSite Templating DCRs)
- `tt_deletedcr` (deleting TeamSite Templating DCRs)
- `all` (all possible values above)

role_list element

The example in this section describes how the `role_list` element works.

The following code means if the user is logged in as a Master (that is, users with an entry in the `master.uid` file), do not display the associated workflow template. All other roles can access the template.

```
<role_list>
  <role value="master" include="no" allusers="no"/>
  <role value="all" include="yes" allusers="no"/>
</role_list>
```

The `allusers="no"` qualifies the fact that all other roles are allowed but you should have a valid user from the <user_list> element.

In the case where <role value="all" include="yes" allusers="yes"/> then even if you appear invalid as defined by the `user` element, the `role` element has precedence over the <user_list> element.

user_list element

The examples in this section describe how the `user_list` element works.

The following code means if the user is Joe, display the associated template, otherwise do not display it.

```
<user_list>
     <user value="joe" include="yes"/>
     <user value="all" include="no"/>
</user_list>
```

The following code means if the user is Joe or Jane, do not display the associated template. If it is any other user also do not display it (this is added by default, as defined in `available_template.dtd`).

```
<user_list>
     <user value="joe" include="no"/>
     <user value="jane" include="no"/>
</user_list>
```

The following code means if the user is Joe, do not display the associated template. If it is any other user, display it.

```
<user_list>
     <user value="joe" include="no"/>
     <user value="all" include="yes"/>
</user_list>
```

branch_list element

The `branch_list` is similar to the `user_list` element, except that it has no dependencies with `role_list` (this is also true of the `command_list`). The example in this section describe how the `branch_list` element works.

Consider a user Jerome, logged in to the TeamSite GUI as described below, attempting a `submit` command.

```
role = "master";
user = "jerome";
branch = "/default/products";
command = "submit";
```

The following `available_templates.cfg` file is processed by the `available_templates.ipl` program (the processing sequence and results of the request are described following the file on page 107).

```
<available_templates>
 <template_file active="yes" name="Custom" path="wfb/custom_submit.wft"/>
     <command_list>
        <command include="no"value="new_job"/>
        <command include="yes"value="all"/>
     </command_list>
     <role_list>
        <role allusers="no"include="yes"value="administrators"/>
        <role allusers="no"include="no"value="authors"/>
        <role allusers="yes"include="yes"value="masters"/>
        <role allusers="no"include="no"value="editors"/>
     </role_list>
     <user_list>
        <user include="yes"value="jerome"/>
        <user include="no"value="all"/>
     </user_list>
     <branch_list>
        <branch value="all"include ="yes"/>
     </branch_list>
 </template_file>
</available_templates>
```

The `available_templates.ipl` program checks the header of the `available_templates.cfg` to find the associated DTD file. In this case, the DTD specified is `available_templates.dtd`.

1. The `available_templates.ipl` program checks the `active` attribute and proceeds because the value is `"yes"`.
2. It then checks the `command_list` element.

   The command rules in this example state to display the template for all types except `new_job`, therefore, since the example input is `command="submit"`, it returns `"true"`.
3. It then checks the `roles_list` element.

   The roles rules specify that:

   - Some administrators are allowed (if they are listed in the `user_list` element)
   - No authors are allowed
   - All masters are allowed
   - No editors are allowed.

   This rule, when applied to the role of master in this example, returns `"true"`.

   Note: The `available_templates.ipl` program only checks the `user_list` element for roles in which `allusers="no"`.
4. Lastly, the `branch_list` element is checked.

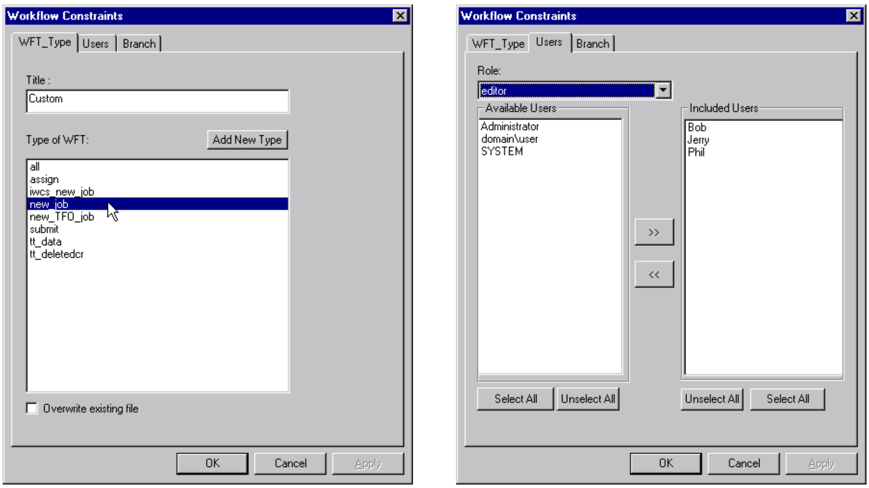
   In this example, all branches are allowed. Therefore the rules return `"true"`, and the template is displayed to the user Jerome.

The `available_templates.ipl` file is explained in detail on page 110.

Modifying available_templates.cfg from WorkflowBuilder

The excerpt from the `available_templates.cfg` file on the next page illustrates a number of points about the file's structure and contents. It includes one of the default workflow templates (`author_submit.wft`) and a template created using WorkflowBuilder called `custom.wft`.

The constraints placed on the `custom.wft` file are called out on page 109 and WorkflowBuilder's Workflow Constraints settings used to create them are shown below.



WFT_Type and Users Tabs of the Workflow Constraints dialog box

Note the following:

- The title Custom is entered in the **Title** field of the **WFT_Type** tab. This title and the actual file name (`custom.wft`) both get referenced in the `available_templates.cfg` file.
- new_job is specified as the template type on the **WFT_Type** tab.
- The **Users** tab's **Role** field is set to editor.
- Three users have been moved to the **Included Users** field. These three users must have been previously added to the `editor.uid` file located in `iw-home/TeamSite/conf/roles`.
- No branch constraints were defined on the **Branch** tab.

Examine the file on the next page to see how the data entered in the Workflow Constraints dialog box are stored in the `available_templates.cfg` file.

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE available_templates SYSTEM './available_templates.dtd'>

<available_templates>
  <template_file name='Author Submit Workflow'
     path='default/author_submit.wft'>
    <command_list>
      <command value='submit' />
      <command value='all' include='no' />
    </command_list>
    <role_list>
      <role value='author' include='yes' allusers='yes'/>
      <role value='all' include='no'  allusers='yes'/>
    </role_list>
  </template_file>
        •
        •
        •
<template_file active="yes" name="Custom"
 path="wfb/custom.wft">
        <command_list>
            <command include="yes" value="new_job">
            </command>
        </command_list>
        <role_list>
            <role include="no" value="admin">
            </role>
            <role include="no" value="author">
            </role>
            <role include="no" value="master">
            </role>
            <role include="yes" value="editor">
            </role>
        </role_list>
        <user_list>
            <user include="yes" value="Bob">
            </user>
            <user include="yes" value="Jerry">
            </user>
            <user include="yes" value="Phil">
            </user>
        </user_list>
    </template_file>
</available_templates>
```

Specifies the dtd file used by this file

Begins a template file section (this is one of the default templates)

Specifies this workflow template can be invoked through Submit

Specifies this workflow template cannot be invoked by other means

Specifies Authors can use this workflow template

Specifies other roles cannot use this workflow template

Ends a template file section

Begins a template file section (this was created using WFB and stored in the wfb directory)

Specifies this workflow template can be invoked through New Job

Specifies this workflow template can only be invoked by Editors

Further specifies that this workflow template can only be invoked by the Editors Bob, Jerry, or Phil

The available_templates.ipl file

The available_templates.ipl file consists of a single Perl function that parses the available_templates.cfg file. It compares the constraints (for example, user or branch) defined in the available_templates.cfg file with the request being made in the TeamSite GUI. It then decides whether or not to list a template in the TeamSite GUI.

The PERL function processes requests based on the following four input parameters:

- The command being issue (for example, new_job, submit, data_tt).
- The branch to which the user is logged on.
- The user that is requesting the list.
- The role the user has used to log into Teamsite in this session.

This enables you to create code that states: if command X is issued by user U who's Role is Master, and the Branch ID is B, then display templates A, B, C, and D. For example, if a New Job command is issued by the user Joe, who is logged in with a Role of Master, display the Author Approval and Editor Approval workflow templates.

Note: Do not edit the available_templates.ipl file.

The available_templates.dtd File

The available_templates.cfg file begins with the following prolog:

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE available_templates SYSTEM './available_templates.dtd'>
```

It declares that the available_templates.cfg uses the available_templates.dtd to describe the expected document structure. The available_templates.dtd file is a collection of declarations divided into two types:

- ELEMENTS—Defines an element and what it can contain.
- ATTLIST—Defines the attributes that are allowed for an element.

By default, the `available_templates.dtd` file is installed in the same directory as the `available_templates.cfg`, either:

- `C:\Program Files\Interwoven\TeamSite\local\config\wft` (Windows servers)
- `iw-home/local/config/wft` (UNIX servers)

The `available_templates.dtd` file defines the default behavior of the `available_templates.cfg`. The file is shown on page 111.

```
<!ELEMENT available_templates (template_file)* >
<!ELEMENT template_file
((command_list)?,(role_list)?,(user_list)?,(branch_list)? >
<!ELEMENT path EMPTY>

<!ELEMENT command_list (command)+ >
<!ELEMENT role_list (role)+ >
<!ELEMENT user_list (user)+ >
<!ELEMENT branch_list (branch)+ >

<!ELEMENT command EMPTY>
<!ELEMENT role EMPTY>
<!ELEMENT user EMPTY>
<!ELEMENT branch EMPTY>

<!ATTLIST template_file
     name CDATA #IMPLIED
     active (yes|no) "yes"
     path CDATA #IMPLIED>

<!ATTLIST command
     value  CDATA "all"
     include (yes|no) "yes">

<!ATTLIST role
     value  (author | admin | master | editor | all) "all"
     include (yes|no) "yes"
     allusers (yes|no) "no">

<!ATTLIST user
     value CDATA "all"
     include (yes|no) "yes">

<!ATTLIST branch
     value CDATA "all"
     include (yes|no) "yes">
```

The iw.cfg File

The `iw.cfg` file is the main TeamSite server configuration file. It includes configuration settings for the way TeamSite looks and responds to various requests. By default, the file is located in `/etc`. This section includes information about workflow-related settings in three parts of the `iw.cfg` file:

- `[iwserver]`
- `[iwsend_mail]`
- `[workflow]`

For details about TeamSite configuration issues that do not concern workflow, refer to the *TeamSite Administration Guide* for your server platform (Windows or Solaris).

[iwsend_mail] Parameters

The Perl script `iwsend_mail.ipl` was specifically designed for use within TeamSite workflows to simplify the creation of external task scripts for email notification. Modify the `[iwsend_mail]` section of your `iw.cfg` file to include the following lines:

```
[iwsend_mail]
maildomain=interwoven.com
mailserver=mail1.interwoven.com
use_mapping_file=true
email_mapping_file=c:/iw-home/local/config/wft/email_map.cfg
debug_output=c:/tmp/iwsend_mail.log
```

For detailed information about the `iwsend_mail.ipl` script, refer to Appendix A, “The iwsend_mail.ipl Script”

[workflow] Parameters

The `[workflow]` section of `iw.cfg` contains by default three commented parameters and their corresponding default values:

- `external_task_add_filelist=false`
- `wftask_nesting_depth_allowed=3`
- `external_task_retry_wait=1`

Each of these parameters also contains a commented description. To activate any of the parameters, remove the single pound sign (#) that precedes the line. You can also change the default setting. Ensure the double pound signs (##) preceding the description are not removed.

```
[workflow]
## Set 'external_task_add_filelist' to false if you want to prevent
## TeamSite from adding files to the command line of external task
## command callouts (recommended on WinNT/2K).  Defaults to true.
#external_task_add_filelist=false

## The maximum depth of nesting allowed for nested jobs (wftask);
## defaults to 3.  Values less than 1 are ignored.
#wftask_nesting_depth_allowed=3

## Set external_task_retry_wait to the number of minutes you want the
## workflow engine to wait before it re-attempts to run an external
## task after failing.  Defaults to 1 minute.
#external_task_retry_wait=1
```

Chapter 7

Workflow Template Files

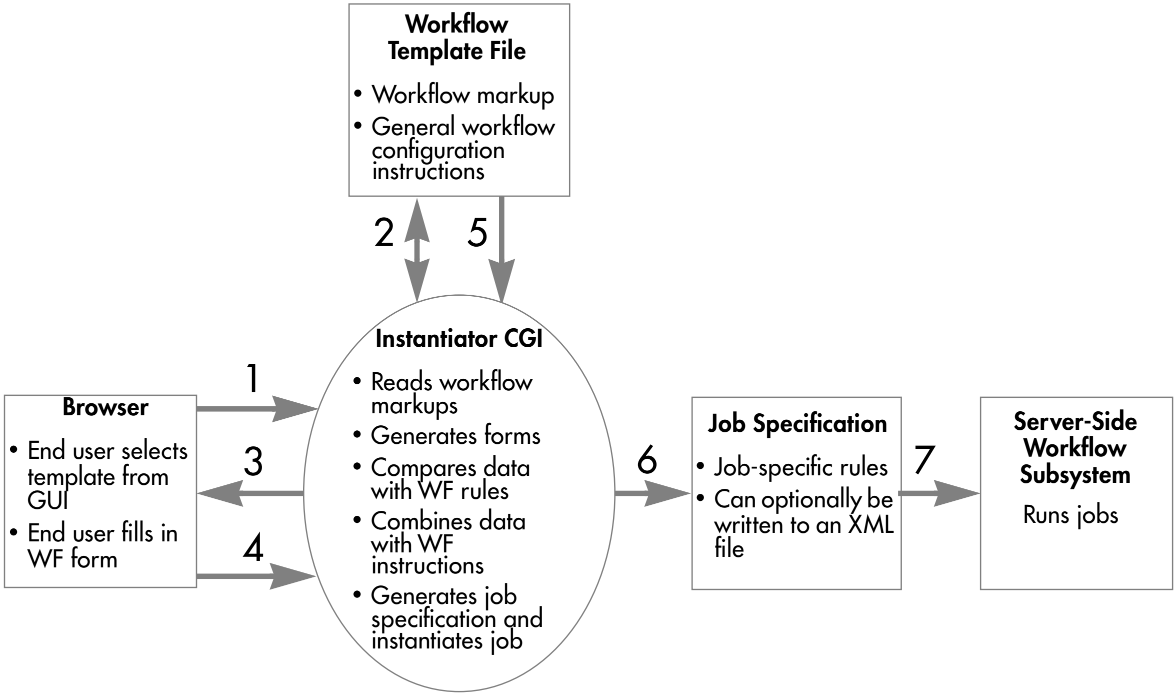
This chapter describes the structure and contents of workflow template files. These files include the sample workflow templates installed by the TeamSite installation program, example files installed by the WorkflowBuilder installation program, and workflow templates created with WorkflowBuilder.

Workflow Illustrated

This manual's Introduction opens with a simplified diagram of the workflow lifecycle (page 14). It is designed to introduce basic workflow concepts. This section includes a detailed illustration that focuses primarily on the server processes and the interaction with the end-user (job creators and authors). It introduces the Instantiator CGI and depicts the four other main components involved in using workflow templates to create jobs:

- **Workflow template**—Defines the workflow rules through a set of workflow markups and a set of general workflow configuration instructions. Workflow templates are typically created using WorkflowBuilder as described in Chapter 4.
- **Instantiator CGI**—Interprets the workflow rules and data from end users, produces browser graphics and prompts, generates a job specification, and instantiates the job.
- **TeamSite browser-based GUI**—Displays forms that prompt end-users for input.
- **Job specification file**—Generated by the instantiator CGI.
- **Server-side workflow subsystem**—Provides a framework for controlling processes involved with these.

The following diagram shows how these components work together. Sections after the diagram explain each diagram step and component in detail.



Workflow Template Overview

Diagram Key

1. In the TeamSite GUI, an end user selects a workflow template from the **File > New File**, **File > Edit File**, or **File > Create New Job** menu item. The instantiator CGI reads the file `available_templates.cfg` to determine which workflow template files are available for that given TeamSite area or file content.

2. The instantiator CGI goes to the specified workflow template file and reads the workflow markup, which consists of Perl instructions residing in the workflow template file's `<template_script>` elements. See page 122 for details about `<template_script>` syntax and usage.

3. Based on the workflow markup, the instantiator CGI creates one or more workflow forms into which an end user can enter workflow configuration information using the TeamSite browser-based GUI.

4. An end-user using the TeamSite GUI enters information in the workflow form and submits it back to the instantiator CGI.

5. The instantiator CGI consults the rules in the workflow template file's workflow markup to verify the validity of the data entered by the end user. If the data meets all necessary criteria, it is parsed by the instantiator CGI (see Step 6). If the data does not meet all necessary criteria, the interface re-prompts the end user so that data can be re-entered (default notification is the invalid field turning red in the workflow form).

6. After determining that the workflow form contains valid data, the instantiator CGI combines the data with the general instructions from the workflow template file to create a job specification (and optionally a job specification file) for this specific job. If a job specification file is created, it is equivalent to the file you would create manually if you defined a job as described in Chapter 9.

   When a job specification file is not created (which is typically the case), the instantiator CGI performs the functional equivalent of writing a job specification file to disk and then invoking the `iwjobc` and `iwinvokejob` commands to instantiate and execute the job instance.

   For an explanation of workflow template file structure and supported element syntax, see "Workflow Template File Structure" on page 119. For an example of a job specification file, see "Sample Job Specification File" on page 195

7. The job is instantiated on the server and started. These are actions you would execute manually (via `iwjobc` and `iwinvokejob`) as described in "Running Manually Created Jobs" on page 173.

The following sections provide more details about each diagram component.

Workflow Template File

A workflow template file is an XML file that can contain any or all of the elements that are valid in a job specification file. These elements form the set of general workflow configuration instructions shown in the diagram on page 116. See "Workflow Template File Structure" on page 119 for details about these elements.

In addition, the workflow template file can contain `<template_script>` elements and a set of directives to define the workflow markups also shown in the diagram on page 116. All instructions residing within a `<template_script>` element are interpreted by the instantiator CGI as Perl code. See “Workflow Template File Structure” on page 119 for details and a sample file illustrating these concepts.

Instantiator CGI

TeamSite includes a standard instantiator CGI, `iwwft_instantiator.cgi`, to perform the following tasks:

- Create and display the workflow information form based on information in the workflow template file.
- Evaluate data entered by end users based on the workflow rules in the workflow template file.
- Combine user-entered data with general workflow configuration instructions to create a job specification.
- Instantiate the job specification on the TeamSite server and start the job.

Browser Interface (GUI)

The browser-based GUI is a standard, TeamSite supported GUI through which end-users can enter data in a workflow form.

Job Specification File

For an explanation of file structure and supported element syntax, see “Job Specification File Structure” on page 175. See “Sample Job Specification File” on page 195 for a sample job specification file.

Server-Side Workflow Subsystem

The server-side workflow subsystem resides on the TeamSite server and contains all the executable files that provide workflow functionality.

Workflow Template File Structure

Workflow template files by default reside in `iw-home/local/config/wft` and end with a `.wft` extension. Workflow template files may contain the following components:

- `<template_script>` elements containing arbitrary Perl code.
- `CGI_info` directives to control the look and feel of workflow forms generated by the instantiator CGI.
- `TAG_info` directives to control the data collection, validation, and error messages displayed in workflow forms.
- `__ELEM__(tagname);` directives to return the number of elements in a tag.
- `__TAG__(tagname);` directives to insert the HTML-encoded data associated with the form POST/GET key *tagname* into the job specification.
- `__INSERT__(string);` directives to insert text into a job specification programatically.
- `__VALUE__(tagname);` directives to return unescaped POST/GET data associated with `$tagname`.
- Other elements identical to those used by job specification files.

The following section contains an excerpt from a basic sample workflow template file, followed by explanations of all file components (some of which are not included in the basic sample file). A second, more sophisticated sample file (“Complex Workflow Template File” on page 137) shows how to use more of these file components.

Simple Workflow Template File

The following is a fragment from a simple workflow template file that fills in blank fields (indicated by __TAG__ directives) with HTML-encoded CGI data.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE workflow SYSTEM "iwwf.dtd">

<template_script><![CDATA[

  TAG_info(

     description  => "<input type='text' value='dark and stormy night'>",
     job_name     => "<input type='text' value='edit story'>",

  );

]]></template_script>

  <workflow name="__TAG__('job_name');"
            owner="__TAG__('iw_areaowner');"
            description="__TAG__('description');">
```

Things to note in the preceding example:

- POST/GET data keynames appear on the left hand side of the arrow in the TAG_info directive.
- The HTML form input field that collects data for the template is located to the right of the arrow in the TAG_info directive.
- The TAG_info directive is located within a <template_script> element.
- You can refer to POST/GET data that was not explicitly collected by the HTML form input fields you specified in TAG_info. For example, iw_areaowner is provided by default, so you do not need to give the template instantiator CGI an input field HTML fragment for iw_areaname within the TAG_info directive.

Suppose that, in the user interface for this form, you want the field that collects data for `job_name` to have a label of "Job Name" instead of "job_name". The following template file section would accomplish that:

```
TAG_info(
     description  => "<input type='text' value='dark and stormy night'>",

     job_name     => [ html  => "<input type='text' value='edit story'>",
                        label => "Job Name",
                       ],
);
```

This example illustrates the `TAG_info` attributes `html` and `label`. There are many more, but all of them follow the same simple pattern:

```
[    ...some_attribute...     =>   ...a_value...,
     ...another_attribute...  =>   ...another_value...,
... and so on...
],
```

As described later in this chapter, the template developer can do far more sophisticated things than just filling in the blanks. For example, you can generate workflow dynamically, and intersperse dynamically generated workflow, data, and tags with hard-coded information. The following sections explain the details of workflow template file components and how you can use them to create workflow templates ranging from simple to elaborate. For another example of how to use many of these component, see "Complex Workflow Template File" on page 137 .

The <template_script> Element

A workflow template file can contain any number of `<template_script>` elements. Each `<template_script>` element contains arbitrary Perl code that can perform the following actions:

- Define the rules that the instantiator CGI employs to combine user-entered data with:
  - hardcoded workflow XML from the workflow template file.
  - programatically generated workflow XML produced within `<template_script>` elements.
- Programatically generate a job specification (and/or intersperse hard-coded XML job specification information with programatically generated XML).
- Optionally send the job specification to a file in addition to, or instead of, instantiating it into the workflow subsystem. This can be helpful if you need to see exactly what XML is being produced by the workflow template file and instantiator CGI.
- Define the rules that validate user-entered data.
- Define the custom error messages displayed when the template's data validation rules are not satisfied (see "The TAG_info Directive" on page 125).
- Inspect incoming POST/GET data, and (under certain conditions) execute code on the basis of this data. For example, rules in a `<template_script>` element can tell the instantiator CGI to generate different job specifications depending on what a user's name is.

  For example, if the an Author "Andre" needs three approvers for his work, but the Author "Jerome" needs only one approver, you can define rules in a `<template_script>` element to perform this job customization automatically based on whether Jerome or Andre is the Author.
- Merge POST/GET data with the hard-coded workflow XML from the workflow template file.
- Determine the look and feel of the automatically generated workflow form that collects end-user data for a job (see "The CGI_info Directive" on page 124).

Because TeamSite workflow templates must be valid XML documents, all content in a <template_script> element must be declared as CDATA to protect it from interpretation by an XML parser. For example:

```
<template_script><![CDATA[
   # arbitrary Perl code
]]></template_script>
```

Together, all of the <template_script> elements in a workflow template file form a single Perl program. If a workflow template file contains more than one <template_script> element, all variables, functions, and libraries set in one element are available in the others. For example:

```
<...hard-coded workflow XML...>

<template_script><![CDATA[
     use Lib1;    # you can import libraries
     sub some_function      # you can define functions
     {
        return "Please enter beverage choice";
     }
     my $beverage = "tea";  # you can define variables
]]></template_script>
<...hard-coded workflow XML...>
<template_script><![CDATA[
     # The variable $beverage is accessible in this
     # section, and contains the value "tea".
     # The function some_function() may also be called here.
]]></template_script>
```

The CGI_info Directive

Usage:

`CGI_info( ...list of key/value pairs... );`

Description:

Sets various defaults that effect the look and feel of workflow forms generated by the instantiator CGI. The `CGI_info` directive may only appear within a `<template_script>` element. Properties that you can set are described in the following table:

Property	Description
`error_data_bgcolor`	Data field background color displayed if an end user enters invalid data (validity is determined by the instantiator CGI). By default, all non-empty fields are valid, but you can customize this on a field-by-field basis. Color in this property and all other color properties shown in this table can be specified using standard HTML color syntax (for example, `"red"`, `"green"`, `"#FFFFFF"`).
`error_label_bgcolor`	Label field background color displayed if an end user enters invalid data in the data field.
`error_text_color`	Error message text color.
`valid_bgcolor`	Background color displayed if an end user enters valid data.
`title`	Browser window title.
`html_body_options`	Sets the options for the `<body>` element of the instantiator CGI. For general information about `<body>` elements, see `http://www.w3.org/TR/REC-html40/struct/global.html#edef-BODY`
`tag_table_options`	Sets the options for the `<table>` element of the instantiator CGI. For general information about `<table>` elements, see `http://www.w3.org/TR/REC-html40/struct/tables.html#h-11.2.1`
`pre_tagtable_html`	Defines what is displayed in a workflow form between the banner and tag table areas.
`post_tagtable_html`	Defines what is displayed in a workflow form after the tag table area.

Note: TeamSite comes with a set of standard defaults to govern the look and feel of workflow forms.

Example:

```
CGI_info(
 error_bgcolor      => "red",
 valid_bgcolor      => "green",
 title              => "TeamSite Workflow Template",
 html_body_options  => "bgcolor='yellow'",
 tag_table_options  => "border=5 cellspacing=2 cellpadding=8",

 pre_tagtable_html  => "<h2>Whatever you want...</h2>",
 post_tagtable_html => "...this appears after the tagtable...",
);
```

The TAG_info Directive

Usage:

`TAG_info(`*`list of key/value pairs`*`);`

Description:

Establishes a relationship between a list of tag names and the information the instantiator CGI uses to collect data for them. There are two ways to build these associations:

Style 1 (simple):

```
tagname =>  "...html that collects data for tagname...";
```

Style 2 (highly flexible):

```
tagname =>  [ html          => "...html that collects data for tagname...",
              is_required   => "true",
              valid_input   => "...Perl expression...",
              label         => "...html label...",
              error_msg     => "...html error message...",
            ];
```

When the instantiator CGI processes the TAG_info directive, the name attribute in the resulting HTML code is automatically set to *tagname*. For example, given the following TAG_info directive:

```
TAG_info(
     beverage => "<input type='text' value='tea'>",
);
```

The internal representation of the resulting HTML code is:

```
"<input type='text' name='beverage' value='tea'>"
```

Because this is done automatically, it is impossible for the tag names to get out of sync with the resulting HTML code. For example, if you attempted to explicitly set the name attribute to something other than *tagname*:

```
TAG_info(
  beverage => "<input type='text' name='drink' value='tea'>",
};
```

then name='drink' gets removed and automatically replaced by name='beverage'.

The TAG_info directive may appear only within a <template_script> element. While it is legal to have any number of TAG_info directives in a workflow template file, it is often convenient to consolidate all necessary data into one TAG_info directive.

Properties that you can set for each tag in a TAG_info directive are described in the following table:

Property	Description
html	Valid HTML input form field (which optionally may contain a default value). This is required if you are using Style 2.
is_required	Whether end-user input in the tag is required. Defaults to true if not set.
valid_input	Rules to check input validity. Default is to check for an empty string if not set.
error_msg	An HTML message returned if end-user input is invalid. Default message is Valid input required if not set.
label	The HTML label displayed beside the HTML input field for this tag. Defaults to the value of *tagname* if not set.

Array Validators

When validating input in a `valid_input` expression, both `$_` and `@_` are set appropriately. Therefore, when collecting information in a multi-select list, a tag's validator can be implemented as follows:

```
TAG_info(

a_tag_name => [ html  => "html that collects data for a_tag_name...",
          valid_input => 'a_tag_validator(@_)',
              ]
)
```

Example:

The following example shows definitions for three tags (named `food`, `beverage`, and `music`). Each tag can be used any number of times by the instantiator CGI to prompt for and collect end-user input in a workflow form.

The definition for tag `food` specifies that the HTML element used to collect data for this CGI variable is a text field.

The tag `beverage` has the following characteristics:

- It only accepts text input.
- It automatically displays a default value of `Beverage: tea` in its entry field.
- A value in its entry field, either end-user input or the default, is required.
- The label `Enter beverage choice` is displayed beside the text field that collects user input.
- `valid_input` specifies that all data entered by an end user must begin with the string `Beverage:`.
- `error_msg` specifies the error message to be displayed if end-user input does not begin with `Beverage:`.

The tag `music` displays a default value of `Punk`.

```
TAG_info(
  food => "<input type='text'>",
  beverage => [ html         => "<input type='text' value='Beverage: tea'>",
                is_required  => 'true',
                label        => '<strong>Enter beverage choice</strong>',
                valid_input  => '/^Beverage:/',
                error_msg    => '<br><strong>'.
                                'ERROR: input must begin with "Beverage:"'.
                                '</strong>',
              ],
  music => "<input type='text' value='Punk'>",
);
```

The __ELEM__ Directive

Usage:

`__ELEM__($tagname);`

Description:

Returns the number of data elements associated with tag *`tagname`*. If *`tagname`* is undefined, 0 is returned. The `__ELEM__` directive may appear inside and/or outside of a <template_script> element. You can also embed an `__ELEM__` directive within an `__INSERT__` directive. A workflow template file can contain any number of `__ELEM__` directives.

Example:

The following `TAG_info` directive defines the tag `reviewers` to accept multiple selections. Therefore, this one tag can have multiple values. By default, two reviewers (Bob and Jerry) have been selected. If an end user accepts these default values, `__ELEM__('reviewers');` will yield `2`. If an end-user also selects Phil as a reviewer, `__ELEM__('reviewers');` will yield `3`.

```
TAG_info(
 reviewers  =>  [  html  => "<select multiple>"            .
                             "      <option>Phil"            .
                             "      <option selected>Bob"   .
                             "      <option selected>Jerry"  .
                             "</select>",
                    label => "Pick reviewers",
                 ],
);
```

The __TAG__ Directive

Usage:

`__TAG__($tagname);`

Description:

When the instantiator CGI creates a job specification, it uses each `__TAG__` directive in the workflow template file as an insertion point for the HTML-encoded value associated with the form input key *`tagname`*. Thus, user input from any tag can be inserted at any point in a job specification file in HTML-encoded form.

In addition, the `__TAG__` directives can mention form input key names that are not defined in `TAG_info` as long as the POST/GET data is provided for these keys programatically. The following POST/GET keys are always passed, and are therefore always available for use in a workflow template file or job specification file. The set of passed tags differs depending on how the job is started as shown in the following tables.

If started by **Submit**:

Key Name	Description
iw_areaowner	The owner of the workarea
iw_branch	The branch's vpath (/default/main/subranch1)
iw_home	The iw-home directory
iw_role	The user's role
iw_session	The session string
iw_template_file	The template file's path and name relative to *iw-home*/local/config/wft
iw_template_name	The template name to be displayed in TeamSite GUI
iw_use_default	Use the default argument of the template (defaults to true)
iw_user	The user's name
iw_workarea	The workarea's vpath (/default/main/WORKAREA/user1)

If started by **New Job**:

Key Name	Description
iw_home	The iw-home directory
iw_role	The user's role
iw_session	The session string
iw_template_file	The template file's path and name relative to *iw-home*/local/config/wft
iw_template_name	The template name to be displayed in TeamSite GUI
iw_use_default	Use the default argument of the template (defaults to true)
iw_user	The user's name

Additionally, the iw_overwrite POST/GET key makes the status of the Overwrite button in the TeamSite GUI available to the workflow subsystem. For example, if Overwrite is selected, an iw_overwrite value of true is passed as POST/GET data to the instantiator CGI, making it available for use in a job specification. If Overwrite is not selected, the value of iw_overwrite is false.

Additional POST/GET keys could also be available, depending on the job's configuration. To display a list of all POST/GET keys available in a specific job, run `show_env.cgi` by naming it in the job's `<cgitask>` element. For example:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE workflow SYSTEM "iwwf.dtd">

     <workflow name="minimal"
               owner="jon" creator="jon"
               description="This is a minimal example of a CGI task">
        <cgitask name="cgi" start="t" owner="chris"
                 description="First CGI task" immediate="t">
           <areavpath v="/default/main/wfregr2/WORKAREA/chris"/>
           <successors>
              <successorset description="Success">
                 <succ v="end"/>
              </successorset>
           </successors>
           <command v="show_env.cgi"/>
        </cgitask>
        <endtask name="end"/>
     </workflow>
```

The `__TAG__` directive may appear inside or outside of a `<template_script>` element. You can also embed a `__TAG__` directive within an `__INSERT__` directive. A workflow template file can contain any number of `__TAG__` directives. To determine how many elements a tag contains, refer to the `__ELEM__` directive. See "Using Variables in Strings" on page 135 for details about the syntax of variables used within the `__TAG__` directive.

Examples:

If your workflow template file contained the following text:

```
I wish I had a __TAG__('beverage');
```

and the instantiator CGI POST/GET data for tag `beverage` was `cup of tea`, the job specification would contain:

```
I wish I had a cup of tea
```

Similarly, if `beverage` were an array tag (for example a multi-select or checkbox), and `2` were a valid index, the following would be a valid entry in the workflow template file:

```
I wish I had a __TAG__('beverage[2]');
```

In this case, the third element from tag `beverage` would be inserted by `__TAG__` . The third element is chosen because arrays start at element 0.

The __INSERT__ Directive

Usage:

`__INSERT__($string);`

Description:

Inserts the value of the variable (or hard coded string) `$string` into the workflow template file, where `$string` can be any arbitrary text (typically, workflow XML). `$string` can optionally include embedded tags (using the `__TAG__` directive) and/or elements (using the `__ELEM__` directive). Embedding tags within an `__INSERT__` directive is especially useful when the template's output needs to be generated dynamically. See "Using Variables in Strings" on page 135 for details about the syntax of variables used within the `__INSERT__` directive.

Example:

The following example shows the portion of a workflow template file that sequentially inserts the values of tags `a`, `b`, and `c` into the job specification file.

```
<template_script><![CDATA[

 my $i;
    my @tag_array = ('a','b','c');
    for ($i=0; $i<3; ++$i)
    {
        __INSERT__("I am __TAG__($tag_array[$i]); pleased!\n");
    }

]]></template_script>
```

Note that an `__INSERT__` directive can also process complex expressions both inside and outside of a <template_script> element (for example, it can process quoted fragments containing nested `__TAG__(...);` directives, possibly joined by '`.`').

The __VALUE__ Directive

Usage:

`__VALUE__(`*`$tagname`*`);`

`__VALUE__(`*`$tagname,$encoding`*`);`

Description:

By default, returns the unescaped POST/GET data associated with tag *`$tagname`*, but unlike `__TAG__(`*`$tagname`*`)`, it does not insert anything into the job specification when the instantiator CGI processes the workflow template file. If the value of the optional parameter *`$encoding`* is set to `html`, the HTML-encoded version of the data is returned instead of the raw value.

This is useful when the template's output needs to be generated dynamically based on the POST/GET values the instantiator CGI receives. The values returned by the variable `$tagname` are as follows:

- If `$tagname` does not refer to a defined POST/GET key name, `undef` is returned.
- If `$tagname` is a scalar POST/GET key name, a scalar is returned.
- If `$tagname` is an array POST/GET key name, a list is returned.
- If `$tagname` is a subscripted array POST/GET key name, a scalar is returned.

The `__VALUE__` directive may only appear within a `<template_script>` element. In a `__VALUE__($tagname);` directive, if the tagname is a subscripted array, the subscript can be an expression.

Example:

The following example uses `__VALUE__` of tag `x` to set the upper limit of `$i`. This example assumes that the form input key name `x` contains an integer.

```
<template_script><![CDATA[

  for (my $i=0; $i < __VALUE__("x");  ++$i)
  {
     __INSERT__("very nice $i\n");
  }

  # Advanced users:  if x were an array tag (CGI form input keyname),
  # then it could be subscripted as follows, assuming 2 is a valid
  # array index (cf: __ELEM__):
  #
  #    for (my $i=0; $i < __VALUE__("x[2]");  ++$i)
  #

]]></template_script>
```

Other Elements

A workflow template file can also contain any element that is legal in a job specification file. These elements, described in "Workflow Template File Structure" on page 119, make up the set of general workflow configuration instructions shown in the workflow template file box in the diagram on page 116.

Using Variables in Strings

The following scenarios describe syntax rules that apply to variables in strings used by `__TAG__` and `__INSERT__` directives. The scenarios start with the simplest method of variable substitution and end with the most advanced.

Scenario 1: Basic Variable Usage

When inside a quoted string, the argument for a `__TAG__` directive does not need any kind of quoting at all.

For example, assuming you have a POST argument named `color1`, you can just say:

```
__INSERT__("shirtcolor='__TAG__(color1);' accepted!!");
```

Other valid usage examples are:

```
__INSERT__("...  __TAG__('color1'); ...");
__INSERT__('...  __TAG__("color1"); ...');
__INSERT__("...  __TAG__(color1); ...");
__INSERT__('...  __TAG__(color1); ...');
__INSERT__("... '__TAG__(color1);' ...");
__INSERT__('... "__TAG__(color1);" ...');
__TAG__("color1");
__TAG__('color1');
```

The following is **not** valid because the argument `color1` is not quoted in any way, and `__TAG__` is not nested within an `__INSERT__` directive:

```
__TAG__(color1);
```

Scenario 2: Including Quotation Marks in Insertions

Continuing with the preceding example and adding the following information:

- you have a Perl variable named `$var1` whose value is `workarea`
- a POST input key named `workarea` whose value is `jon`

then the following statements all insert the string `jon` into the job:

```
__INSERT__("... __TAG__($var1); ...");
__INSERT__("... __TAG__('$var1'); ...");
__TAG__($var1);
```

The following expression inserts the string `'jon'` into the job:

```
__INSERT__("... '__TAG__($var1);' ...");
```

Therefore, to insert a tag into a job within single quotes you could say:

```
__INSERT__("var1='__TAG__(color1);' accepted!!");
```

And to insert a tag into a job within double quotes, you could say:

```
__INSERT__('var1="__TAG__(color1);" accepted!!');
```

Scenario 3: Preferred Ordering of Single and Double Quotes

If you specify either of the following:

```
__INSERT__('__TAG__("$var1");');
__INSERT__('__TAG__($var1);');
```

you will probably get an error message about not finding data for the FORM input keyname `$var1` because the outer-most quotation marks on the `__INSERT__` directive are single quotes. In Perl, single quotes are interpreted as:

`“Do not interpolate anything in this string as a Perl variable.”`

Hence `$var1` is literally the set of characters `$`,`v`,`a`,`r`,`1` (and not a variable named `$var1` whose value is `workarea`).

In general you should place the double quotes outside and the single quotes inside:

```
__INSERT__("var1='__TAG__(color1);' accepted!!");
```

For example:

```
<template_script><![CDATA[

     my $status = "not in stock.";
         __INSERT__("var1='__TAG__(color1);' currently $status");

]]></template_script>
```

Complex Workflow Template File

The following workflow template file is more elaborate than the sample file shown in "Simple Workflow Template File" on page 120; it shows the use of several additional file components as explained in the preceding sections. Specifically, this file:

- Generates a form that captures the deployment date for this job.
- Ensures that the "Timed Deployment" field does not allow a user to just enter `"later"`.
- Sets the label in the form that collects data for the deploy date to "`Timed Deployment`".
- Sets the owner attribute for the XML element `<workflow>` to the HTML-encoded data associated with the form input key `iw_areaowner` (and similar operations for the other `__TAG__` directives).
- For each file that has been selected in the TeamSite GUI, it inserts a line that reads:

  ```
  <file path='...filename...' comment='File to time deploy'/>
  ```

  Note: When the job specification is generated, these lines appear between the XML tags `<files>` and `</files>`.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE workflow SYSTEM "iwwf.dtd">

<template_script><![CDATA[

 TAG_info(
    deploy_date => [ html          => "<input type='text' value=''>",
                     valid_input => '$_ ne "later"',
                     label       => "Timed Deployment",
                   ],
 );

]]></template_script>

<workflow name="TimedDeploy" owner="__TAG__('iw_areaowner');"
          creator="__TAG__('iw_areaowner');"
          description="Timed Deployment">

 <usertask name="AuthorWork" owner="__TAG__('iw_areaowner');"
    description="Editing files to time deploy" start="t">

      <areavpath v="__TAG__('iw_workarea');"/>
      <successors>
      <successorset description="One Minute">
      <succ v="Submit"/>
      </successorset>
      </successors>
      <files>
      <template_script><![CDATA[

       for (my $i=0; $i < __ELEM__('iw_file'); ++$i)
        {
          __INSERT__("<file path='__TAG__(iw_file[$i]);' comment='File to
                          time deploy'/>\n");
        }

      ]]></template_script>
      </files>
 </usertask>

 <submittask name="Submit" owner="__TAG__('iw_areaowner');"
    description="Staging for deployment.">
    <areavpath v="__TAG__('iw_workarea');"/>
       .
       .
       .
```

Debugging Workflow Files

Two additional POST/GET keys are available for debugging workflow template and job specification files. Details are as follows.

iw_debug_mode

The `iw_debug_mode` key instructs the instantiator CGI to process input data from a submitted form as it normally would, and then display job-specific information in a Debug Mode page rather than instantiate the job on the server. The Debug Mode form always contains two default sections: the Perl code (including line numbers) that generates the job specification, and the XML job specification itself. This job specification is what would have been instantiated on the server if debug mode had been turned off. A third section showing syntax errors appears in the Debug Mode form if the instantiator CGI found errors in the Perl code it generated based on form input.

iw_output_file

The `iw_output_file` key instructs the instantiator CGI to process input data from a submitted form as it normally would, and then capture the output in an XML job specification file rather than instantiate the job on the server. After it is created, you can manually instantiate the job specification file on the server at any time via `iwjobc`.

Usage:

You can define the `iw_debug_mode` and `iw_output_file` key names in a `TAG_info` directive (causing the keys to appear in the workflow form), or you can provide definitions programmatically via POST/GET data.

Example:

The following example shows definitions that are set within a `TAG_info` directive:

```
TAG_info(
     iw_debug_mode  => "<input type='text' value='true'>",
     iw_output_file => "<input type='text' value='/tmp/my_job.xml'>",
);
```

Things to note in the preceding example:

- For either element, setting `type` to `text`, the element appears on the workflow form. Setting `type` to `hidden` causes the element not to be displayed on the workflow form.
- You can toggle debug mode on or off by setting `value` to `true` or `false`.
- The file named in `value` must be writable by `httpd`.

Workflow Log File

Output from workflow runtime diagnostics is logged in:

- *iw-home*`\local\logs\iwjoberrors.log` (Windows)
- `/var/adm/iwjoberrors.log` (UNIX)

Sample Workflow Templates

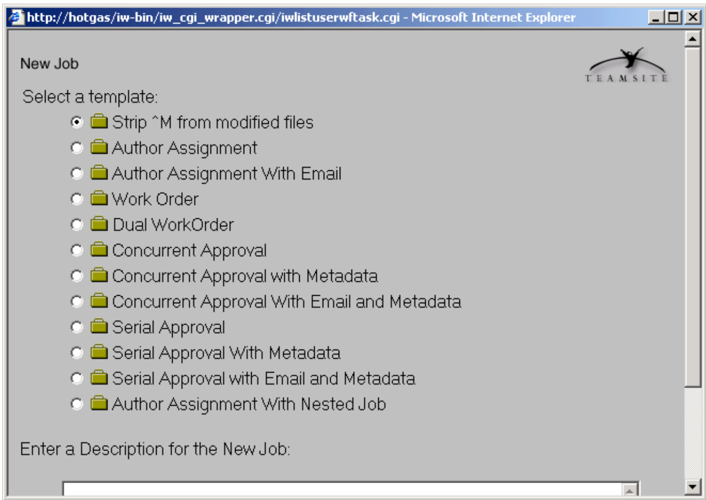
The TeamSite installation program installs sample workflow templates for several common jobs. Some of these templates are available by default from the TeamSite GUI, and others require that you configure TeamSite to make them available. The following sections describe where each template resides, the job that is configured by each template, and how to make the template available through the TeamSite GUI.

Sample Template Locations

Sample workflow templates reside in three directories:

- *iw-home*`/local/config/wft/default`
- *iw-home*`/local/config/wft/examples`
- *iw-home*`/local/config/wft/solutions`

Following a TeamSite installation, templates in the `default` directory are automatically displayed as choices in the New Job window when a user selects **File > New Job** in the TeamSite GUI.



Templates in the `examples` and solutions directories require that you configure the `available.templates.ipl` file to make them available from the TeamSite GUI. After you make them available, they appear together with the default templates in the New Job window. The following sections describe the default and example templates in detail.

Default Template Descriptions

The following workflow templates are installed in `iw-home/local/config/wft/default`:

Template Name	Description
`author_assignment.wft`	Lets an Editor, Administrator, or Master assign a job to an Author. The assigner selects an author and enters a task description. The assigner also selects a branch and workarea if the job is initiated from the To Do List view. An approval sequence is also included for the author assignment.
`author_assignment_with_email.wft`	Lets Editors, Administrators, and Masters assign a job to an Author. The workflow notifies the Author via email.

Template Name	Description
author_submit.wft	Submits a data content record (DCR) to the staging area when an Author clicks **Save and Exit** in TeamSite Templating Data Content Record window. This automates the submission process, eliminating the need for the Author to initiate the submission manually after creating or editing a DCR.Requires the presence of TeamSite Templating.
default_submit.wft	Performs the same actions as the **Submit Direct** button, plus provides support for pre-submit activities including approval, file type recognition, and user-specific destinations.
dual_work_order.wft	Configures a job that lets an Editor, Administrator, or Master assign to another user the task of setting up the job defined by the Work Order template. Requires additional configuration as described in "Configuring the Dual Work Order Template" on page 145.
work_order.wft	Configures a template that lets an Editor, Administrator, or Master define work assignments and approval chains for a job of any length. Requires additional configuration as described in "Configuring the Work Order Template" on page 142.

By default, these workflow templates are referenced in the `available_templates.cfg` file as described in "The available_templates.cfg File" on page 102. Therefore, all are available via the **File** > **New Job** command following a TeamSite installation.

Note: The `work_order` and `dual_work_order` templates are highly configurable. It is recommended that you configure TeamSite as described in the following sections before allowing end-users to run the Work Order and Dual Work Order jobs.

Configuring the Work Order Template

The Work Order template (`work_order.wft`) describes a job that enables an Editor, Administrator, or Master define work assignments and approval chains for a job of any length. The job can contain:

- Single or multiple contributors
- Single or multiple approvers
- Serial contributors and approvers

- Concurrent contributors and approvers
- Metadata assignments
- Task attribute assignments (for example, whether files are locked or read-only)
- Recursion (jobs that run again at specified intervals of time)
- Archiving (saving the job for a specified amount of time following its completion)

After you perform the configuration tasks described in this section, the Work Order job is ready for use by job creators. A job creator running the job can then enter data in the Work Order template using the TeamSite GUI to control the following parameters for the current instance of the job:

- Which users will be content contributors
- Which users will be content approvers
- Which users will receive email notification when a task is done
- Whether contributors and approvers will work serially or concurrently
- Whether files are locked
- Whether files are read-only
- Whether contributors can add metadata to a file
- How many days will elapse before the job runs again
- How long the job will be saved after it is completed

To configure the Work Order template:

1. Open the `work_order.wft` in a text editor.
2. Set the `$number_of_contributors` variable as described in the file's comments to specify how many times the contributor field appears in the Work Order form.

   The default value is `2`.
3. Set the `$number_of_approvers` variable as described in the file's comments to specify how many times the approver field appears in the Work Order form.

   The default value is `2`.

4. Set the `@possible_contributors` variable as described in the file's comments to specify which TeamSite roles will be used as the basis for the drop-down list of possible contributors.

   The default roles are Author and Editor.

5. Set the `@possible_approvers` variable as described in the file's comments to specify which TeamSite roles will be used as the basis for the drop-down list of possible approvers.

   The default role is Master.

6. Set the `$skip_metadata` variable as described in the file's comments to specify whether the Work Order form will contain a metadata field.

   The default value is `FALSE`, which creates a metadata field in the Work Order form.

   If a metadata field exists in the form, the person using the form to set up the job instance can specify whether content contributors will be prompted to set metadata for a file.

7. Set the `$skip_email` variable as described in the file's comments to specify whether the Work Order form will contain a field for email addresses.

   The default value is `FALSE`, which creates an email address field in the Work Order form.

   If an email address field exists in the form, the person using the form to set up the job instance can specify who receives email notification upon task assignment.

8. Set the `$skip_save_job` variable as described in the file's comments to specify whether, and for how long, the job is saved following completion.

   The default value is `FALSE`, which creates a save job field in the Work Order form.

   If a save job field exists in the form, the person using the form to set up the job instance can specify the duration (in days) that the job is saved.

9. Set the `$skip_recurring` variable as described in the file's comments to specify whether, and how often, the job reoccurs.

   The default value is `FALSE`, which creates a job recursion field in the Work Order form.

   If a job recursion field exists in the form, the person using the form to set up the job instance can specify how many days elapse before this job runs again.

10. Set the `$skip_branch` variable as described in the file's comments to specify whether the person filling in the Work Order form is prompted for branch and path information.

    The default is `FALSE`.

11. Optionally, set default values for any of the following variables:

Variable	Current Default Value
`$workflow_name`	`Ordered Change Request`
`$contributor_default`	No default
`$task_desc_default`	`Do this`
`$contrib_email_default`	No default
`$contrib_perform_tasks_default`	`serial`
`$metadata_default`	`no`
`$contrib_lock_default`	`yes`
`$approver_default`	No default
`$approver_email_default`	No default
`$approver_perform_tasks_default`	`serial`
`$approver_lock_default`	`no`
`$approver_read_only_default`	`yes`
`$workarea_path_default`	No default
`$recurring_days`	No default
`$numrows_default`	`1`

    A line already exists for each variable's default in `work_order.wft`. To set no default for a variable, set its value to `""` (an empty string). Do not comment out any of the variables. See the comments in `work_order.wft` for more information.

Configuring the Dual Work Order Template

The Dual Work Order (`dual_work_order.wft`) template describes a job that lets an Editor, Administrator, or Master assign to another user the task of setting up the job defined by the Work Order template.

For example, a second-level manager could use the Dual Work Order template to delegate a job's setup to a first-level manager. The second-level manager would fill in the initial Dual Work Order form, stating which first-level manager should complete the job setup. When the second-level manager starts the job, the first-level manager receives the job setup assignment and the Dual Work Order template automatically starts the Work Order job as an external task for the first-level manager to complete. At completion of the entire job, the second-level manager can approve the entire job before it is submitted.

Note: The Dual Work Order job runs the Work Order job as an external task, ensure that `work_order.wft` is configured as described in the previous section before running Dual Work Order.

To configure the Dual Work Order template:

1. Open the `dual_work_order.wft` in a text editor.
2. Set the `$areavpath` variable so that it specifies a path to any workarea on the system.

   This step is necessary so that the CGI task that runs as part of the Dual Work Order job is directed to TeamSite. Therefore, you can specify any valid TeamSite workarea when setting `$areavpath`.
3. Set the `$number_of_contributors` variable as described in the file's comments to specify how many times the contributor field appears in the Dual Work Order form.

   The default value is `2`.
4. Set the `@possible_contributors` variable as described in the file's comments to specify which TeamSite role(s) will be used as the basis for the drop-down list of possible contributors.

   The default roles are Author and Editor.
5. Optionally, set default values for any of the following variables:

Variable	Current Default Value
`$workflow_name`	`Coordinated Change Request`
`$contributor_default`	No default
`$task_desc_default`	`Do this`

Variable	Current Default Value
`$contrib_email_default`	No default
`$self_approve_default`	`yes`
`$self_email_default`	No default
`$numrows_default`	`1`

A line already exists for each variable's default in `dual_work_order.wft`. To set no default for a variable, set its value to `""` (an empty string). Do not comment out any of the variables. See the comments in `dual_work_order.wft` for more information.

Example Template Descriptions

The templates in `iw-home/local/config/wft/examples` are included as reference examples that are applicable to some installations. The functionality provided by these examples is included in a more generalized form in the `work_order.wft` template. The templates in the `examples` directory are provided as shorter, more modular examples of how you can develop custom workflow templates.

To make the example files available to end-users, you must edit `available_templates.cfg` as described beginning on page 102.

The following example templates reside in `iw-home/local/config/wft/examples`:

Template Name	Description
`author_assignment_with_nested_job.wft`	Lets Editors, Administrators, and Masters assign a job that contains two tasks, the second of which does not begin until the first has been approved by an editor.
`concurrent_approval.wft`	Lets Editors, Administrators, and Masters assign a task to a content contributor and specify a single user or group as the approver.
`concurrent_approval_with_email_with_metadata.wft`	Performs the same activities as Concurrent Approval, plus allows the content contributor to set metadata for a file, and sends email to the approver(s).

Template Name	Description
concurrent_approval_with_ metadata.wft	Performs the same activities as Concurrent Approval, plus allows the content contributor to set metadata for a file.
serial_approval.wft	Lets Editors, Administrators, and Masters assign a task to a content contributor and specify one or more users as the approvers.
serial_approval_with_email_ with_metadata.wft	Performs the same activities as Serial Approval, plus allows the content contributor to set metadata for a file, and sends email to the approvers.
serial_approval_with_ metadata.wft	Performs the same activities as Serial Approval, plus allows the content contributor to set metadata for a file.

You should examine each `.wft` file for details about its construction and the features of the job it defines. After examining each file, you can choose to use it as is, or modify it for your specific installation using the information from “Workflow Template File Structure” on page 119.

Regular Expression Support

You can use regular expressions within workflow templates to search for a specified pattern and specify what to do when matching patterns are found. For example, you can perform regular expression searches for constraints on workflow templates. Consider the following TeamSite structure:

```
/default/main ──┐
                │
                ├─ administrator ──┬a1
                │                  ├a2
                │                  └a3
                ├─ marketing ──────┬m1
                │                  ├m2
                │                  └m3
                │
                └─ sales ──────────┬s1
                                   ├s2
                                   └s3
```

If you want only the users in the three `administration_1` branches (`a1`, `a2`, and `a3`) to access a workflow template, you can set the following constraint:

```
<branch_list>
 <branch value="/default/main/administrator_1/a1" include="yes" />
 <branch value="/default/main/administrator_1/a2" include="yes" />
 <branch value="/default/main/administrator_1/a3" include="yes" />
 <branch value="all" include="no" />
</branch_list>
```

If a new branch called `a4` is added to `/default/main/adminstrator_1` you could manually update the `available templates.cfg` file to allow access for users in the new branch by adding the following line:

```
 <branch value="/default/main/adminstrator_1/a4" include="yes" />
```

Or you could modify the `available templates.cfg` file to use the following regular expression and automate the constraints placed on the `a4` branch:

```
<branch_list>
 <branch value="/deault/main/adminstrator_1/.*" include="yes" />
 <branch value="all" include="no" />
</branch_list>
```

This regular expression allows users from any branch under `/deault/main/adminstrator_1` to have access to the template.

Chapter 8

Solutions Workflows

TeamSite 5.5.2 includes six new *solutions* workflows. These workflows were designed to incorporate functionality that was commonly being implemented by modifying the Interwoven-provided workflow template files (`wft`).

Three of these workflows are variations on the `author_submit` workflow included with this and earlier TeamSite releases. The `author_submit` workflow has been extended to include email notifications, metadata capture, and file deployment using OpenDeploy. These workflows are:

- `author_submit_with_deploy.wft`
- `author_submit_with_email.wft`
- `author_submit_with_metadata.wft`

The other three new workflows are configurable alternatives to the previous default workflows (`default_author_submit.wft`, `default_submit.wft` and `author_assignment.wft`) included with this and earlier versions of TeamSite. These configurable workflows are:

- `configurable_author_submit.wft`
- `configurable_default_submit.wft`
- `configurable_author_assignment.wft`

Each of the configurable workflows has a number of configurable options specified in an external configuration file called *`workflow_name`*`.cfg`. For example, the `configurable_author_submit.wft` file has a corresponding configuration file called `configurable_author_submit.cfg`. Each configurable workflow also has a corresponding properties file called *`workflow_name`*`.properties`. These configuration files are described in “Workflow-specific Configuration Files” on page 152 and the properties files in “Workflow Properties File Overview” on page 153.

The six new workflows (and associated configuration files) are installed by the TeamSite 5.5.2 installation program in a directory called *iw-home*/local/config/wft/solutions.

Note: None of the six new workflows are active when the TeamSite 5.5.2 is installed. To activate any of them, you must complete the procedure described in "Enabling Solution Workflows" on page 160.

Workflow-specific Configuration Files

This section introduces the workflow-specific configuration files. For detailed information about a specific cfg or properties file, refer to the "Solutions Workflows Illustrated" on page 153 for an overview of a specific workflow or "Configurable Workflow Settings" on page 164 for details about each configuration option.

Workflow cfg File Overview

Each *workflow_name*.cfg file contains entries for a number of optional features that can be defined as on (*feature*=yes) or off (*feature*=no) without having to edit the wft file. The configuration of a feature (for example, email notification) is identical in each workflow where it is included. The following list contains the configurable features that are available in each of the cfg files.

configurable_author_submit.cfg

- Deployment
- Email Notification
- Metadata Capture
- Review

configurable_default_submit.cfg

- Deployment
- Email Notification
- Metadata Capture

configurable_author_assignment.cfg

- Add Files
- Deployment
- Email Notification
- Metadata Capture
- Review

Workflow Properties File Overview

The *workflow_name*.properties file contains the text strings that are displayed in the Webdesk GUI. These text strings include:

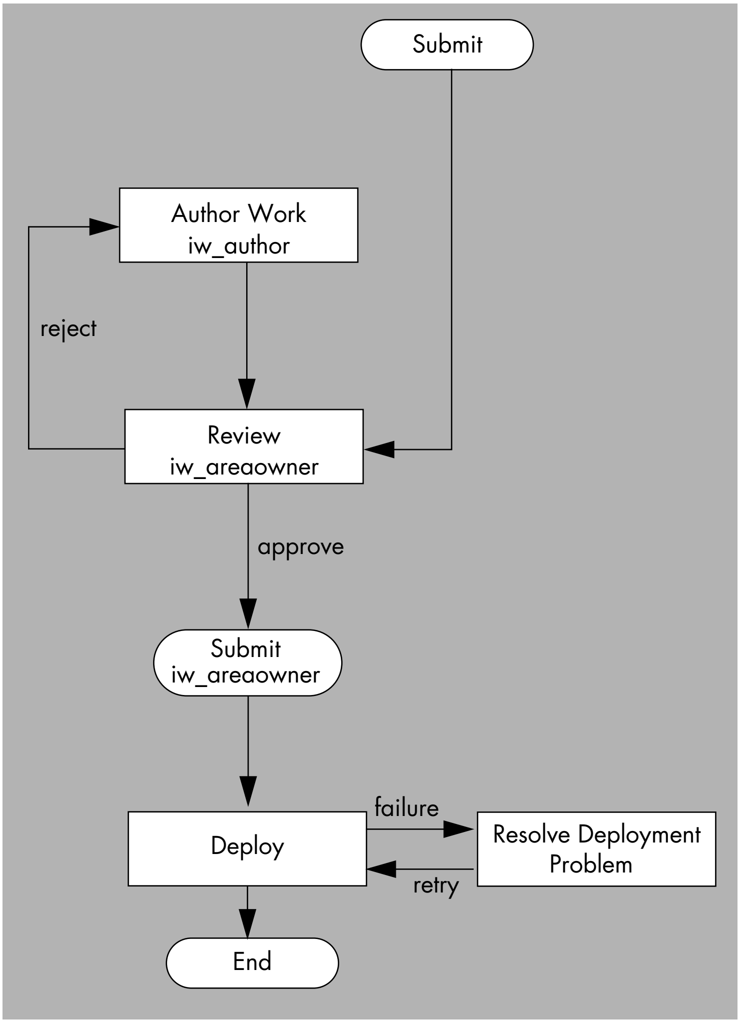
- Field Labels
- Error messages
- Window and dialog box titles
- Image files
- User prompts
- Task names and transition labels

In addition to the *workflow_name*.properties file, there is a *workflow_name_locale*.properties file for each of the supported locales. For more information about these files, see “Configurable Workflows and Localization” on page 170.

Solutions Workflows Illustrated

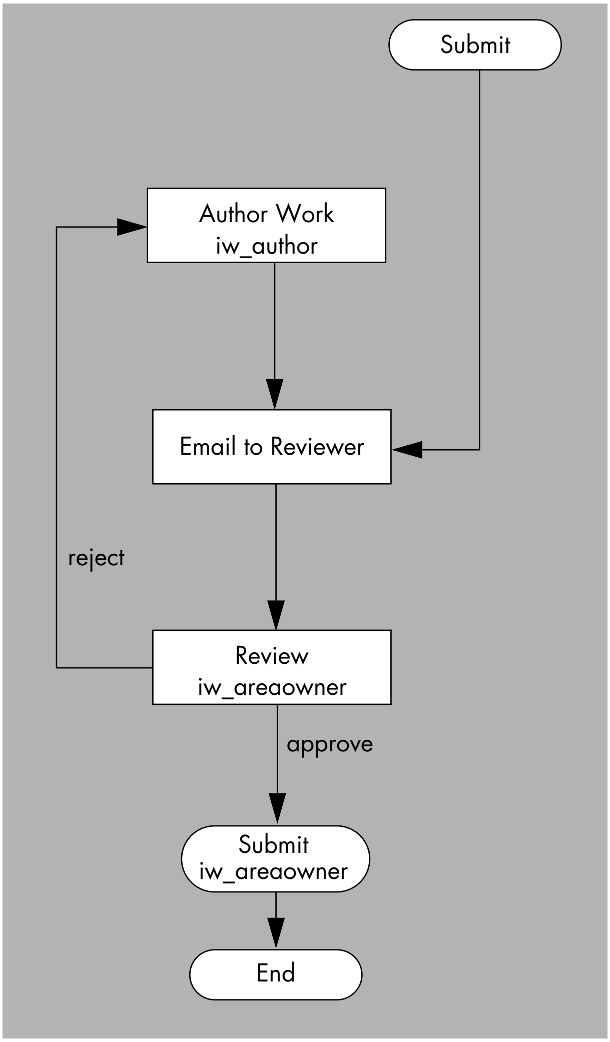
The following sections illustrate the tasks and transitions included in each of the new workflows. A description accompanies each diagram. The configurable steps are marked as “optional” and are described in detail in “Configurable Workflow Settings” on page 164.

Author Submit with Deploy



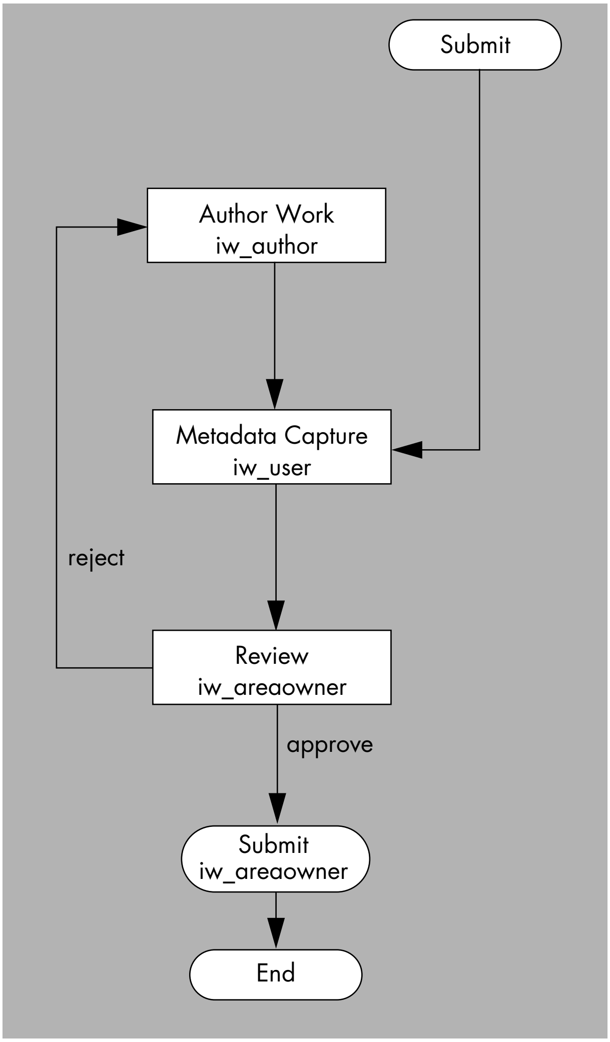
1. Author-submitted content is sent to a reviewer. The reviewer is the defined as the owner of the workarea to which the files are submitted.
2. The reviewer either:
   - Approves the work.
   - Rejects the work (which is sent to an author for modification, and resubmitted for approval).
3. When approved, the file is submitted for deployment.
4. The file is deployed to the location specified in the `wft_opendeploy.cfg` file.

Author Submit with Email



1. Author clicks **Submit** to submit modified files which initiates a new Job and prompts the author for the following information:
   - Submit comment (required)
   - Information comment (optional)
   - Individual file comments (optional)
2. The submitted files trigger an email notification to a reviewer. The reviewer is the defined as the owner of the workarea to which the files are submitted.
3. The reviewer either:
   - Approves the work.
   - Rejects the work (which is sent to an author for modification. When resubmitted, another email is sent to the reviewer).
4. When approved, the files are submitted to STAGING.

Author Submit with Metadata



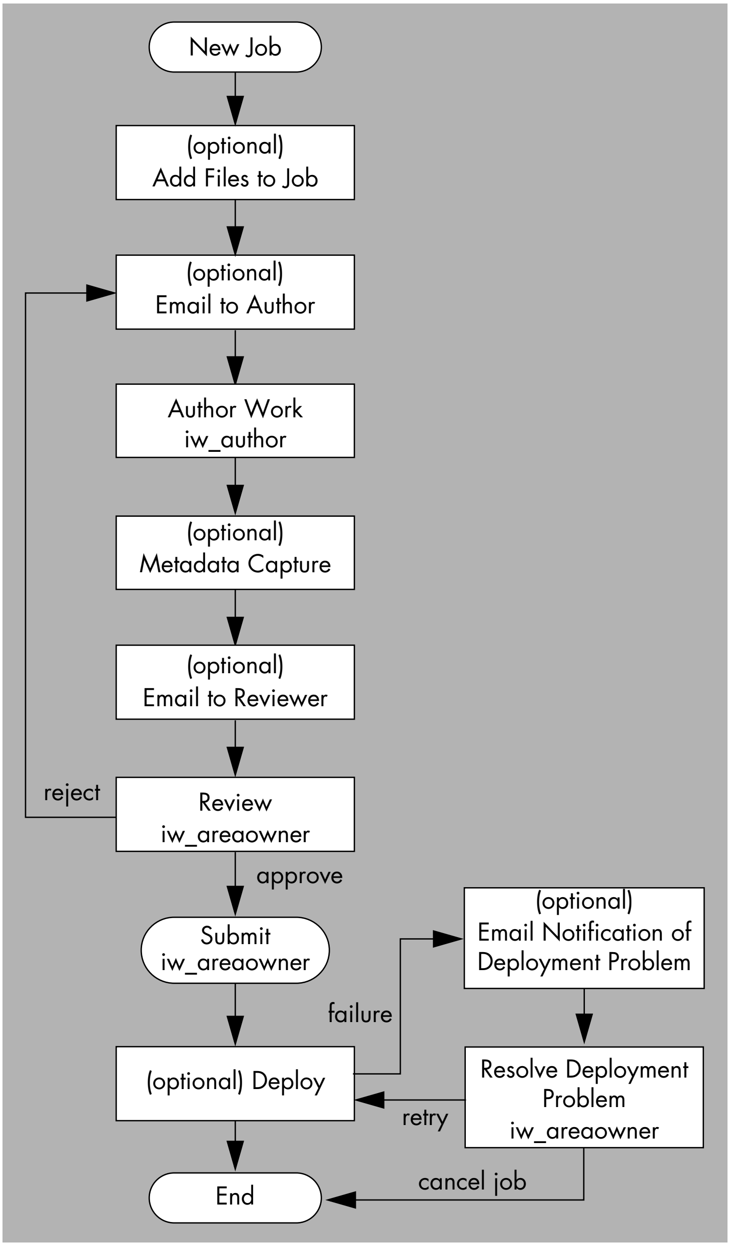
1. Author-submitted content is processed for metadata capture (by either the TeamSite Metadata Capture form, or through MetaTagger 3.1 integration).

2. The file is passed to a reviewer. The reviewer is the defined as the owner of the workarea to which the files are submitted.

3. The reviewer either:
   - Approves the work.
   - Rejects the work (which is sent to an author for modification, then resubmitted for metadata capture and approval).

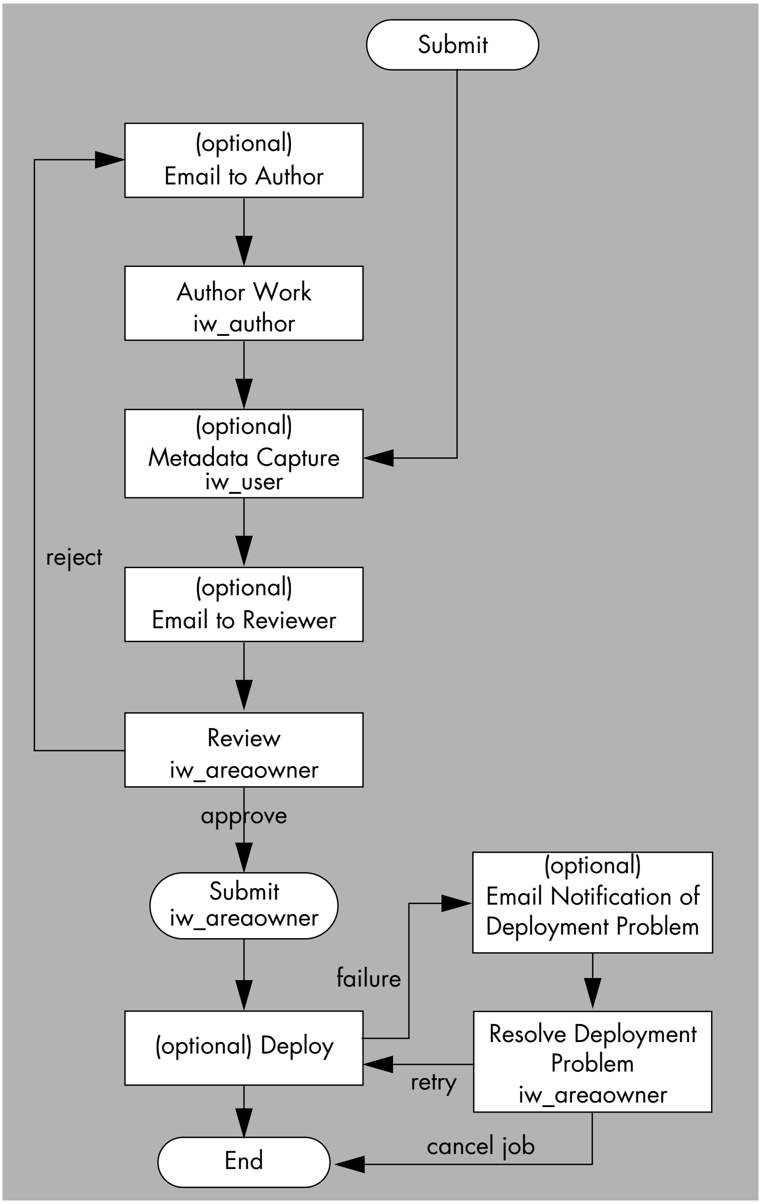
4. When approved, the file is submitted to STAGING.

Configurable Author Assignment



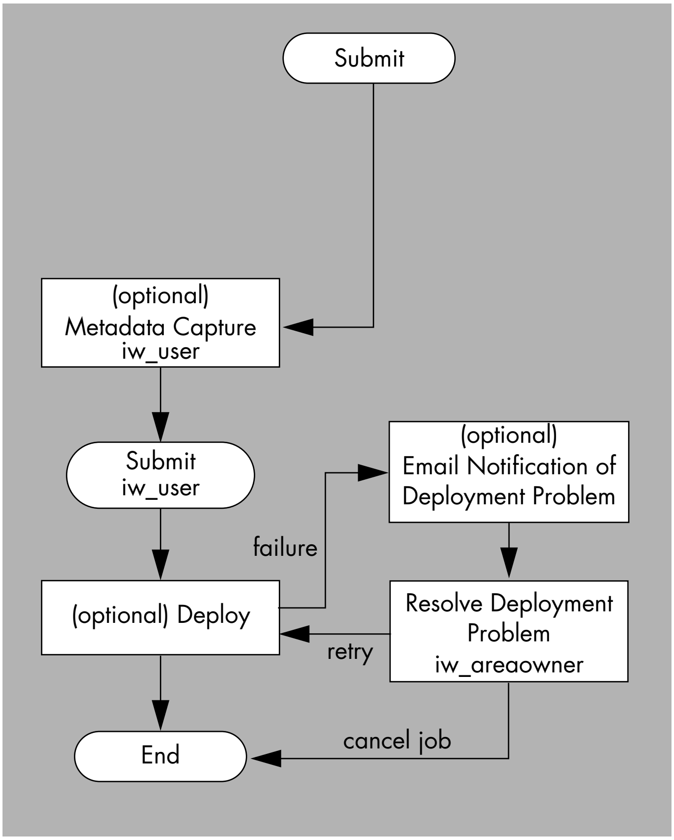
1. A new job is initiated, typically by an editor, the initiator (optionally) selects files to be worked on by the author.
2. (Optional) Email notification of the assigned work and associated files is sent to an author.
3. Author completes assigned work and marks the task as Done.
4. (Optional) Content is processed for metadata capture (by either the TeamSite Metadata Capture form, or through MetaTagger 3.1 integration).
5. (Optional) Email notification is sent to a reviewer configured in the `configurable_author_assignment.cfg` file.
6. The reviewer either:
   - Approves the work.
   - Rejects the work (which triggers email to an author for modification, then resubmitted for metadata capture and approval).
7. When approved, the file is submitted to STAGING.
8. (Optional) The file is deployed to the location specified in the `wft_opendeploy.cfg` file (email is sent to the initiator if there is a deployment problem).

Configurable Author Submit



1. Author-submitted files are (optionally) processed for metadata capture (by either the TeamSite Metadata Capture form, or through MetaTagger 3.1 integration).
2. (Optional) Email notification is sent to a reviewer configured in the `configurable_author_submit.cfg` file.
3. The reviewer either:
   - Approves the work.
   - Rejects the work (which triggers email to an author for modification, then resubmitted for metadata capture and approval).
4. When approved, the file is submitted to STAGING.
5. (Optional) The file is deployed to the location specified in the `wft_opendeploy.cfg` file (email is sent to the initiator if there is a deployment problem).

Configurable Default Submit



1. A user—whose work does not need reviewed—submits a file.
2. (Optional) Content is processed for metadata capture (either in the TeamSite Metadata Capture form, or through MetaTagger 3.1 integration), then submitted to STAGING.
3. (Optional) The file is deployed to the location specified in the `wft_opendeploy.cfg` file (email is sent to the initiator if there is a deployment problem).

Enabling Solution Workflows

The six new workflows are optional and must be activated before using them. Verify that you have satisfied the following list of requirements:

- Install TeamSite 5.5.2 (required)
- Install TeamSite Templating (required)—Workflow email notifications use the presentation template complier installed with TeamSite Templating.
- Install MetaTagger 3.1 (optional)—MetaTagger 3.0 is not supported. If you are integrating with MetaTagger, TeamSite Templating must be installed before MetaTagger or the MetaTagger GUI will not work.
- Install OpenDeploy 5.5.1 or above (optional)—You must have a base server on the TeamSite server.
- The permissions on the `iw-home/tmp` and the `iw-home/tmp/cci` directories must be readable and writable to all TeamSite users (the email notifications are temporarily placed in these directories while being created).

Complete the following procedure to activate any of the new workflows.

1. Open the `iw-home/local/config/wft/available_templates.cfg` file.
2. Add an entry for each new workflow.

   For example, to add the `Author Submit Workflow with Email` workflow, add the following entry:

```
<template_file name='Author Submit Workflow with Email' path='solutions/
author_submit_with_email.wft'>
  <command_list>
     <command value='submit' />
     <command value='all' include='no' />
  </command_list>
  <role_list>
     <role value='author' include='yes' allusers='yes'/>
     <role value='all' include='no' allusers='yes'/>
  </role_list>
</template_file>
```

Note: For your convenience, a file containing entries for each new workflow is provided. It is called `available_templates.cfg.fragment` and is located in the *`iw-home`*`/local/config/wft/solutions` directory. You can copy any or all of the workflow entries from this file into your `available_templates.cfg` file.

3. If you are replacing another workflow, you can deactivate it by any of these methods:
   - Delete the entry
   - Comment out the entry using `<!-- -->`
   - Add the attribute `active="no"`
4. Save and close the `available_templates.cfg` file.
5. For each configurable workflow that you added to your `available_templates.cfg` file, edit the corresponding `.cfg` file to activate the desired functionality.

   The `.cfg` file contains question and answer pairings for each configurable option. For example, in the `Email Notification` section:

   ```
   # Should a email notification be sent if a deploy task fails?
   email_no_deploy=no
   ```

   Change the default from `no` to `yes` on any feature you want to activate.

   Note: For a detailed description of the configurable features available in each workflow, refer to "Configurable Workflow Settings" on page 164.
6. If you are using the `Email Notification` functionality:

   a. Ensure that your `iw.cfg` file contains entries for `maildomain` and `mailserver` in the `[iwsend_mail]` section.

   If it does not, add the appropriate entries. For example:

   ```
   [iwsend_mail]
   maildomain=yourcompany.com
   mailserver=mail.yourcompany.com
   ```

   b. Edit the `solutions/email_map.cfg` file to specify the mapping from usernames to email addresses if adding *`@yourcompany.com`* to the username will not be adequate.

7. If you are using `Metadata Capture` functionality:

   a. Ensure that the `metadata_capture_ui` properties in the workflow-specific `.cfg` files have the desired setting for either MetaTagger or TeamSite Metadata.

   b. Ensure that *iw-home*`/local/config/` contains `datacapture.cfg` and `meta-data-rules.cfg` files, and that these files contain the desired settings.

   c. Ensure the `mt-home/conf/metatagger.cfg` file contains the desired entries.

      See *MetaTagger Administration Guide* "Configuring MetaTagger to Work with TeamSite" for integration instructions.

   d. Ensure that the category `tag` values in the `metatagger.cfg` file match corresponding item `name` values in the `datacapture.cfg` file.

   e. Optionally, test your metadata capture configuration from a custom menu item before you try it from a workflow.

8. If you are using `Deployment` functionality:

   a. Copy the deployment configuration file (`solutions/wft_opendeploy.xml`) to the *od-home*`/conf` directory.

   b. Edit `solutions/wft_opendeploy.cfg` file to specify the mapping from branch names to the corresponding destination nodes and paths.

      See "Integrating with OpenDeploy" on page 162 for detailed information about these entries.

Integrating with OpenDeploy

The TeamSite installation program includes the following files that are required for integration with OpenDeploy:

- `wft_opendeploy.xml`—Deployment configuration file
- `wft_opendeploy.ipl`—Perl script that starts the deployment

- `wft_opendeploy.cfg`—OpenDeploy configuration file which provides a mapping between the branch name and the corresponding deployment attributes (node name and the target directory) required for deployment. Each deployment mapping contains three required items, delimited by commas:
  - Branch name—Name of the source branch.
  - Destination node name—Logical node name of the OpenDeploy receiver that is configured in the source OpenDeploy machine's `*iwodhome*/etc/odnodes.xml` file.
  - Destination path—Path on the destination server where the content will be deployed. This path must be included as an allowed directory in the OpenDeploy receiver's `*iwodhome*/etc/odbase.xml` or `*iwodhome*/etc/odrcvr.xml` configuration file.

These files are installed in *iw-home*`/local/config/wft/solutions`.

Complete the following steps to configure the deployment integration:

1. Copy the `wft_opendeploy.xml` file to the `$ODHOME/conf` directory.
2. Edit the last three lines `wft_opendeploy.cfg` file to reflect the mapping between the branch name containing the file to be deployed and the destination. The default lines are:

```
branchName=/default/main/br3/,dst=/space/vijay/odreceive2,useNode=INTERNET
branchName=/default/main/br2/,dst=/space/vijay/odreceive1,useNode=INTERNET
branchName=/default/main/br1/,dst=/space/vijay/odreceive,useNode=INTERNET
```

source branch destination node name destination path

For example:

`branchName=/default/main/br3/,dst=/wwwroot/target,useNode=INTERNET` (UNIX)
`branchName=/default/main/br3/,dst=\wwwroot\target,useNode=INTERNET` (Windows)

3. Add or delete lines according to the number of source branches and deployment destinations you need to configure.
4. Save and close the file.

Configurable Workflow Settings

All three configurable workflows share the following functionality:

- Email Notification
- Metadata Capture
- Deployment

The `configurable_author_assignment`, and `configurable_author_submit` workflows add the functionality to have submitted files reviewed by a configurable reviewer. The `reviewer` section of these `cfg` files is where the email notification obtains instructions to determine the email address of the reviewer.

The `configurable_author_assignment` workflow also adds the functionality to attach files to a workflow task.

The general procedure for configuring these options is the same for each of the configurable workflows:

1. Open the *`workflow_name`*`.cfg` file that corresponds with the configurable workflow you want to implement.

   By default, these files are located in *`iw-home`*`/local/config/wft/solutions`.
2. Edit the entries that corresponds with the functionality you want to enable.
3. Save and close the file.
4. Ensure the workflow is activated as described in “Enabling Solution Workflows” on page 160.

The sections that follow describe each of these options in detail.

Email Notification Settings

All configurable workflows have the ability to send email notifications, including assignment of work, requests for review of completed work, and problems deploying files.

```
## Email Notification
##
# Should a email notification be sent to an Author when a task
# is pending?
email_notification_to_author=no
```

Change this entry to `email_notification_to_author=yes` if you want to email sent to authors who have had work assigned to them (that is, the status of the task is `pending`). If set to `yes`, email is sent to the author who performs the submit (`author_submit` workflows), or the author to which the work is assigned (`author_assignment` workflows).

```
# Should a email notification be sent to a reviewer when a task
# is pending?
email_notification_to_reviewer=no
```

Change this entry to `email_notification_to_reviewer=yes` if you want to email sent to reviewers who have had work assigned to them for review (that is, the status of the task is `pending`). If set to `yes`, you must configure the reviewer in the `reviewer` section of the `configurable_author_assignment.cfg` or `configurable_author_submit.cfg` files.

```
# Should a email notification be sent if a deploy task fails?
email_no_deploy=no
```

Change this entry to `email_no_deploy=yes` if you want to email sent to the job initiator when a deploy task fails.

```
# Should the initiator be prompted to confirm or change these
# choices when initiating a job?
# If yes, the values of email_notification_to_author,
# email_notification_to_reviewer, and email_no_deploy will serve
# as the defaults.
ask_email_notification_to_author=no
ask_email_notification_to_reviewer=no
ask_email_no_deploy=no
```

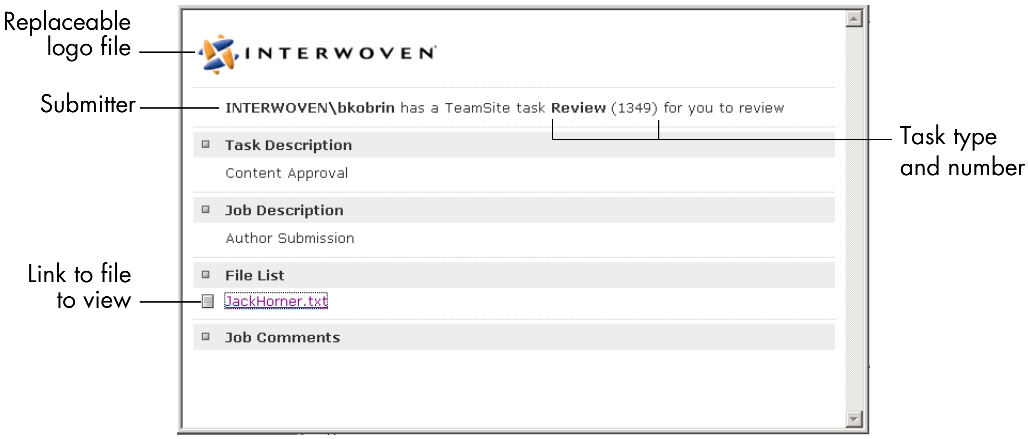
Change this entry to `ask_email_...=yes` if you want the person initiating the job to be prompted to override (on a job-by-job basis) whether email should be sent for any of the three situations.

```
# Templates for the headers and bodies of email messages.
reviewer_email_header_template=reviewer_iwmailheader.tpl
reviewer_email_body_template=reviewer_iwmailbody.tpl
author_email_header_template=author_iwmailheader.tpl
author_email_body_template=author_iwmailbody.tpl
no_deploy_email_header_template=no_deploy_iwmailheader.tpl
no_deploy_email_body_template=no_deploy_iwmailbody.tpl
```

If you want to replace the Interwoven logo file (`ts_logo.gif`) with a graphics file containing your organization's logo:

- Copy the file into *iw-home*`/httpd/iw-icons/solutions` directory.
- Open the appropriate `_mailbody.tpl` file in a text editor.
- Replace the `ts_logo.gif` reference in the `<!-- Logo -->` section with a reference to the new graphic file.

The following graphic shows an example of an email sent to a reviewer:



Metadata Capture Settings

The three *workflow_name*.cfg files contains the following metadata capture entries. They use a question and answer format with the question commented out (preceded by #) followed on the next line by the answer. The answer is set to no by default, and may be followed by additional settings.

```
## Metadata Capture
##
# Should there be a task to capture metadata from the Author?
metadata_capture=no
```

Change this entry to metadata_capture=yes if you want to include a metadata capture task for the corresponding workflow. If set to yes, when a task is mark as Done, the workflow will display either the MetaTagger capture screen (requires that the MetaTagger 3.1 product is installed) or the default TeamSite metadata capture screen depending on a setting later in this file.

```
# Should the initiator be prompted to confirm or change this
# choice when initiating a job?
# If yes, the value of metadata_capture will serve as the default.
ask_metadata_capture=no
```

Change this entry to ask_metadata_capture=yes if you want the person initiating the job to be prompted to override (on a job-by-job basis) whether the submitter is required to enter a metadata term.

```
# If there is a task to capture metadata, which UI should be presented?
# iwmetadata.cgi is the TeamSite metadata capture UI.
metadata_capture_ui=iwmetadata.cgi
# mtmetaproxy.cgi is the MetaTagger metadata capture UI (requires MetaTagger).
#metadata_capture_ui=mtmetaproxy.cgi
```

If a metadata task is included in the workflow, the default metadata capture interface displayed to the submitter is the TeamSite metadata capture form (iwmetadata.cgi). If you have installed MetaTagger 3.1, and would like to have it displayed by the workflow, comment out the metadata_capture_ui=iwmetadata.cgi line (add a # at the beginning of the line) and uncomment the #metadata_capture_ui=mtmetaproxy.cgi line (remove the #).

For more information about using capturing metadata refer to the:

- *TeamSite Administration Guide* for information about iwmetadata.cgi
- MetaTagger user documentation for information about mtmetaproxy.cgi

Deployment Settings

The three *workflow_name*.cfg files contains the following deployment entries. They use a question and answer format with the question commented out (preceded by #) followed on the next line by the answer. The answer is set to no by default, and may be followed by additional settings.

```
## Deployment
##
# Should there be a Deploy task at the end of a job to deploy files
# after they have been submitted?
deploy_task=no
```

Change this entry to deploy_task=yes if you want the files being submitted to be deployed by the workflow using OpenDeploy.

```
# Should the initiator be prompted to confirm or change this
# choice when initiating a job?
# If yes, the value of deploy_task will serve as the default.
ask_deploy_task=no
```

Change this entry to ask_deploy_task=yes if you want the person initiating the job to be prompted to override (on a job-by-job basis) whether the files associated with this job should be deployed when submitted.

```
# Should the job form include an info field?
ask_for_info=yes
```

By default, the job form presented to the submitter includes a field where details about the submission can be included. The information entered by the submitter is displayed as a submit-info variable on the submit task.

Review Settings

The configurable_author_submit.cfg and configurable_author_assignment.cfg workflows contains functionality that requires files be reviewed before they are submitted. The configuration involves defining who the reviewer is. The default reviewer is as follows:

- For configurable_author_submit.cfg—Owner of the workarea to which the file is being submitted.
- For configurable_author_assignment.cfg—Selected Editor

The last line of this section contains the default setting (`configurable_author_submit` is shown here):

```
## Review
##
# Who should review the Author's work?  The format of the review_type is:
#
#  "review_type=" ( "areaowner"|"job_initiator"|<user>|<grouptask>|<select> )
#  <user> ::= "user:" <os_user_name>
#  <grouptask> ::= "grouptask[" (<group>|<user>) ("," (<group>|<user>))* "]"
#  <group> ::= "group:" <os_group_name>
#  <select> ::= "select[" <role> ("," <role>)* "]"
#  <role> ::= "role:" ( "author"|"editor"|"admin"|"master"| etc. )

# For example:
# review_type=areaowner
# review_type=job_initiator
# review_type=user:worldcorp\andre
# review_type=grouptask[group:demomaster,user:jsmith]
# review_type=select[role:editor,role:author]
review_type=areaowner
```

The `configurable_author_submit` (shown here) uses `review_type=areaowner` as its default. To specify one of the following reviewers:

- Job initiator—Uncomment the `# review_type=job_initiator`
- Specified user—Uncomment the `# review_type=user:worldcorp\andre`, and replace `worldcorp\andre` with the username and domain (Windows only).
- Specified group—Uncomment the `# review_type=grouptask[group:demomaster,user:jsmith]`, and replace `demomaster` with a group defined in the operating system. You can also append the group with additional individual reviewers by using the comma-separated `user:`*username* format.
- Selection list from which to choose a reviewer or group of reviewers—Uncomment the `# review_type=select[role:editor,role:author]` to display a menu to the job initiator containing the choice of editor and author.

  Note that the `configurable_author_assignment.cfg` has a default defined as `review_type=select[role:editor]` therefore those users with the role of editor are notified.

Ensure you place a comment (#) in front of the default setting if you select another reviewer setting.

The graphic on page 166 shows an example of an email sent to a reviewer.

Add Files Settings

Only the `configurable_author_assignment.cfg` workflow contains functionality for the initiator to attach files to a job. When a new job is created by selecting **Assign**, **Submit**, or **New Job**, all of the selected files are attached to the job. The configuration involves whether the workflow will begin with a user task for the initiator to add additional files.

```
## Add Files
##
# Should there be an "Add Files" task at the beginning of the job
# to permit additional files to be added?
include_add_files=yes

# Should the initiator be asked if an "Add Files" task is desired
# when initiating a job?
# If yes, the value of include_add_files will serve as the default.
ask_include_add_files=yes
```

Configurable Workflows and Localization

In addition to the *`workflow_name`*`.cfg` file and *`workflow_name`*`.properties` files installed by the TeamSite installation program, there are five additional `properties` files. Four of these files have been localized, and one has not. The unlocalized properties file is used if the user's browser locale does not match any of the localized files.

- *`workflow_name`*`.properties` (default)
- *`workflow_name`*`_de.properties` (German)
- *`workflow_name`*`_en.properties` (English)
- *`workflow_name`*`_fr.properties` (French)
- *`workflow_name`*`_ja.properties` (Japanese)

These files contain the localized text strings that are displayed in the Webdesk GUI if the user's browser is operating in a locale that matches one of these files. If the locale of the browser does *not* match one of the locales, the default `workflow_name.properties` file (the one without a locale specified) is used to provide the text strings.

Note: The contents of these files use Latin-1 encoding, therefore multibyte characters are represented using Unicode characters.

Chapter 9

Job Specification Files

In addition to creating Job files in WorkflowBuilder (as described in Chapter 4, "Using WorkflowBuilder"), you can create a job by directly editing an XML *job specification file*. This job specification file must:

- Reside in the TeamSite home directory
- Be structured as described in "Job Specification File Structure" on page 175
- Use elements as described in "Element Definitions" on page 175

See "Sample Job Specification File" on page 195 for an example of this type of file. See `http://www.xml.com/axml/testaxml.htm` for a detailed XML specification.

Running Manually Created Jobs

After creating a job specification file to define a workflow model, run the job by completing the following procedure:

1. Change to the directory that corresponds with your platform:
   - **`cd iw-home/bin`** (UNIX)
   - **`cd C:\Program Files\Interwoven\TeamSite\bin`** (Windows NT and 2000)
2. Execute `iwjobc` from the command-line to create an instance of the job on your TeamSite server.
3. Execute the `iwinvokejob` utility.

Note: While a job specification file can only define a single workflow model, it is possible to instantiate multiple, identical, concurrent jobs by instantiating and executing the same job specification file more than once using `iwjobc` and `iwinvokejob`. Upon invocation, the job runs until one of its end tasks is activated. Once a job ends, its instance is removed, and you must re-instantiate it to run it again.

Other command-line utilities enable you to destroy jobs, view the state of any object in the workflow system, add files to a particular task within the job, and otherwise interact with running jobs. See *TeamSite Command-Line Tools* for details about the following workflow utilities:

`iwaddtaskfile`	`iwcallback`	`iwgetwfobj`
`iwinvokejob`	`iwjobc`	`iwqueryjobs`
`iwquerytasks`	`iwretrywfop`	`iwrmjob`
`iwrmtaskfile`	`iwtaketask`	`iwtaskselect`
`iwundochoice`		

While the workflow subsystem can be configured to create and save a job specification file for any job, the normal scenario is for the job to be instantiated without the job specification being saved in a file. Saving the job specification in a file is a step that is usually taken only when you need to view the file for debugging.

Job Specification File Structure

A job specification file describes a single job. It is structured as a hierarchy of sections, each containing an element definition that enables you to control a job parameter. An initial `<workflow>` section defines the overall characteristics of the job. It is followed by one or more task sections describing specific tasks that occur as part of the job.

The following list shows all of the possible elements that can define sections in a job specification file. Indentation shows nesting levels:

```
workflow
   usertask
   updatetask
   submittask
   externaltask
   endtask
   grouptask
   cgitask
   dummytask
   locktask
```

All of these elements, their attributes, and their subelements are described in the following section. See “Sample Job Specification File” on page 195 for examples of files that use these elements.

Element Definitions

The following DTD excerpts describe the syntax for each job specification file element. These elements are also valid in a workflow template file.

Note: Subelements within an element must be ordered as shown in the DTD. See `iw-home/local/config/wft/iwwf.dtd` for the complete workflow DTD.

<workflow> Element

A `<workflow>` element defines a job's name and owner.

```
<!ELEMENT workflow (description?, variables?,
(usertask|submittask|updatetask|externaltask|cgitask|endtask|grouptask|
dummytask|locktask)+)>
     <!ATTLIST workflow name ID #REQUIRED
                             owner CDATA #REQUIRED
                             creator CDATA #REQUIRED
                             description CDATA #IMPLIED>
```

Attributes:

`name`	Name of the job. Job names are not unique identifiers. However, each job that is instantiated is identified by a unique ID number.
`owner`	The owner responsible for the job (defined in workflow template file rules).
`creator`	The user who started the job via the TeamSite GUI's workflow form.
`description`	A description of what the job does. Can be specified as both an attribute and a subelement of `<workflow>`.

Subelements:

`<description>`

A description of what the job does. Can be specified as both an attribute and a subelement of `<workflow>`. Syntax is as follows:

```
<!ELEMENT description (#PCDATA)>
```

`<variables>`

Workflow variables are key-value pairs that can be stored in and retrieved from job instances. They are used to allow separate CGI tasks and external tasks to communicate with each other during job execution. Workflow variables are manipulated using the `iwjobvariable` CLT or by specifying them at job creation time. Syntax is as follows:

```
<!ELEMENT variables (variable+)>
   <!ELEMENT variable EMPTY>
      <!ATTLIST variable key NMTOKEN #REQUIRED
                         value CDATA #REQUIRED>
```

Parameters Common to All Tasks

The following parameters apply to all *task* elements (`<usertask>`, `<updatetask>`, `<submittask>`, `<externaltask>`, `<grouptask>`, and `<cgitask>`). In this section, the term *task* represents any of these elements. For information about parameters that apply only to a specific *task* element, see that element's section later in this chapter.

```
<!ELEMENT task (description?, areavpath, successors, timeout?,
                files?, activation?, inactivate?, resets?,
                eastartop*, eafinishop*, variables?)>
    <!ATTLIST task owner CDATA #REQUIRED
                   name ID #REQUIRED
                   start (t|f) "f"
                   description CDATA #IMPLIED
                   lock (t|f) "f"
                   readonly (t|f) "f">
```

Attributes:

`owner`	The owner of the task.
`name`	The name of the task. A task is uniquely identified within its job by its name.
`start`	Specifies whether the task should be active upon its containing job's invocation. The default is `f`.
`description`	A description of what the task does. Can be specified as an attribute as well as a subelement of *task*.

`lock`	When the `lock` attribute is set to `t` the task will acquire TeamSite locks on all the files it contains when it becomes active. If the task cannot acquire locks for one or more of the files it contains it will release any locks it has already acquired and try again every five minutes until it successfully acquires all locks. When a locking task tries to acquire a lock for a file it checks first to see if that file is locked by some other task in its own job. If it is, the locking task "steals" the lock from the other task. This behavior can result in submit time conflicts. In general it is best to ensure that no task will try to acquire locks that could already be owned by another active task.
`readonly`	Marking a task read only disallows users from adding, removing, or modifying files. Note `readonly` is used only by `<usertask>` and `<grouptask>`.

Subelements:

`<areavpath>`

The `<areavpath>` subelement specifies the TeamSite area associated with this task. Syntax is as follows:

```
<!ELEMENT areavpath EMPTY>
    <!ATTLIST areavpath v CDATA #REQUIRED>
```

`<timeout>`

A timeout is an optional time limit for the completion of a task. When time runs out the task is inactivated and the `<succ>` elements are signalled to become active. The time value for `<timeout>` is specified as the `v` attribute in two possible forms: *+HHHHMM*, which is the number of hours and minutes after the task becomes activated that the timeout should occur, or *MMDDYYYYHHMM*, which is the month, day, year, hour, and minute at which the timeout should occur. When using *+HHHHMM*, you must use all six digits, including leading zeros if necessary. Syntax is as follows:

```
<!ELEMENT timeout (succ)+>
    <!ATTLIST timeout v CDATA #REQUIRED>
    <!ELEMENT succ EMPTY>
        <!ATTLIST succ v IDREF #REQUIRED>
```

<files>

These are the files that the actions of a task affect. The files can be specified at configuration time (but only on `<start>` tasks) or dynamically (but only on active tasks). It is expected that the user interface will allow users to modify and/or add to the comment field. Syntax is as follows:

```
<!ELEMENT files (file+)>

<!ELEMENT file EMPTY>
       <!ATTLIST file path CDATA #REQUIRED
                   comment CDATA #REQUIRED>
```

<activation>

The `<activation>` element specifies the conditions under which a task will become active. The body of the `<activation>` element specifies a logical expression. When a finishing task signals a successor task, the successor task notes that the finishing task has signaled and then evaluates the logical expression to determine if it should become active. Syntax is as follows:

```
<!ELEMENT activation (and|or|not|pred)>
<!ELEMENT and (and|or|not|pred)*>
<!ELEMENT or (and|or|not|pred)*>
<!ELEMENT not (and|or|not|pred)>
<!ELEMENT pred EMPTY>
   <!ATTLIST pred v IDREF #REQUIRED>
```

For example, given tasks A, B, C that can signal task D, the `<activation>` element for D looks like this:

```
<activation>
       <and>
              <pred v="A"/>
              <or>
                     <pred v="B"/>
                     <pred v="C"/>
              </or>
       </and>
</activation>
```

This means (in more conventional notation): A & (B | C). When A, B or C signals D, D notes the fact that it has been signaled, and evaluates A & (B | C) where the values of A, B and C are whether they have signalled D. In this example, D becomes active if and only if A has signalled and B or C have signalled.

<inactivate>

A task becomes inactive at the time it signals its successors. However it is often necessary to inactivate tasks other than those which have signalled a task when that task becomes active. For example, suppose a user completes a task and routes it simultaneously to five user for review. If one of those reviewers rejects the work, the task should be inactivated and removed from the lists of the other four reviewers. Syntax is as follows:

```
<!ELEMENT inactivate (pred+)>
```

For example, given tasks A and B that can signal it, task C has the following `<activation>` and `<inactivate>` sections:

```
<activation>
       <or>
              <pred v="A"/>
              <pred v="B"/>
       </or>
</activation>
<inactivate>
       <pred v="A"/>
       <pred v="B"/>
</inactivate>
```

When C becomes active, by being signalled by either A or B, it inactivates both A and B. Specification of the <inactivate> element is optional. If the <inactivate> element is unspecified it is the same as specifying an <inactivate> element containing all possible predecessor tasks.

`<resets>`

A task can be configured to reset the activation state of an arbitrary set of other tasks when it becomes active. Resetting the activation state of a task simply means that such tasks are set to a state of no tasks having activated them. This capability is useful in certain parallel task configurations. Syntax is as follows:

```
<!ELEMENT resets (reset+)>
      <!ELEMENT reset EMPTY>
            <!ATTLIST reset v IDREF #REQUIRED>
```

`<eastartop>`, `<eafinishop>`

Both when a task becomes active (`<eastartop>`) and when a task becomes inactive (`<eafinishop>`), TeamSite extended attributes can be set, modified, or deleted on the files contained by the task. Syntax is as follows:

```
<!ELEMENT eastartop EMPTY>
      <!ATTLIST eastartop op (set|append|delete) #REQUIRED
                             name CDATA #REQUIRED
                             value CDATA #REQUIRED>

<!ELEMENT eafinishop EMPTY>
      <!ATTLIST eafinishop op (set|append|delete) #REQUIRED
                              name CDATA #REQUIRED
                              value CDATA #REQUIRED>
```

If the `op` attribute of the `<ea`*`XXX`*`op>` element is `set`, the extended attribute with key *`name`* will be set to *`value`*. If `op` is `append`, *`value`* will be appended. If `op` is `delete`, the extended attribute with key *`name`* will be deleted.

The *`value`* attribute of the `<ea`*`XXX`*`op>` element can contain the following macros of the form *`%name`*`;` that will be expanded before being set as an extended attribute:

Macro Name	Description
`%workflow;`	Name of the job
`%workflowid;`	ID of the job
`%task;`	Name of the task
`%taskid;`	ID of the task
`%taskowner;`	Owner of the task

Macro Name	Description
%time;	The current wall clock time
%area;	VPATH of the task's area
%path;	Path of the file from area root
%fullpath;	Full path of the file from server root
%taskcomment;	Task-specific comment added to the extended attribute
%filecomment;	File-specific comment added to the extended attribute

<usertask> Element

A <usertask> element defines user tasks that appear on a user's task list.

```
<!ELEMENT usertask (description?, areavpath, successors, timeout?,
                        files?, activation?, inactivate?, resets?,
                        eastartop*, eafinishop*, variables?)>
     <!ATTLIST usertask owner CDATA #REQUIRED
                            name ID #REQUIRED
                            start (t|f) "f"
                            description CDATA #IMPLIED
                            lock (t|f) "f"
                            readonly (t|f) "f">
```

Attributes:

There are no attributes unique to user tasks. See "Parameters Common to All Tasks" on page 177 for descriptions of the attributes shown in the preceding DTD excerpt.

Subelements:

See "Parameters Common to All Tasks" on page 177 for descriptions of the subelements from the preceding DTD excerpt that are common to other tasks. Usage of the <successors> subelement is as follows:

`<successors>`

The `<successors>` subelement specifies the possible alternative sets of successor tasks to signal when the user task is finished. The GUI presents the user with a set of options for finishing a task. The text of the description of the task is used to label the alternatives for the user (for example, “Mark Done”, “Reject”, “Approve” and so on).

```
<!ELEMENT successors (successorset+)>

<!ELEMENT successorset (description?, succ+)>
    <!ATTLIST successorset description CDATA #IMPLIED>

<!ELEMENT succ EMPTY>
    <!ATTLIST succ v IDREF #REQUIRED>
```

<grouptask> Element

A `<grouptask>` element is similar to a user task in that it appears on a user’s task list. A group task, however, belongs to an arbitrary group of users and therefore shows up in the task list of every user who belongs to that arbitrary group. A group task becomes identical in behavior to a user task when one user from the group takes ownership of the task via the GUI or the CLT `iwtaketask`.

```
<!ELEMENT grouptask (description?, areavpath, successors, sharedby,
                        timeout?, files?, activation?, inactivate?,
                        resets?, eastartop*, eafinishop*, variables?)>
    <!ATTLIST grouptask name ID #REQUIRED
                            start (t|f) "f"
                            description CDATA #IMPLIED
                            lock (t|f) "f"
                            readonly (t|f) "f">
```

Attributes:

There are no attributes unique to group tasks. See “Parameters Common to All Tasks” earlier in this chapter for descriptions of the attributes shown in the preceding DTD excerpt.

Subelements:

See “<usertask> Element” on page 182 for a description of the `<successors>` subelement. See “Parameters Common to All Tasks” on page 177 for descriptions of the other subelements from the preceding DTD excerpt that are common to other tasks. Subelements that are unique to group tasks are as follows:

<sharedby> Element

The `<sharedby>` element specifies the arbitrary set of users who share this group task. The element allows an arbitrary combination of individual TeamSite users and OS groups to be shared owners of the group task. Syntax is as follows:

```
<!ELEMENT sharedby (user|group)*>
        <!ELEMENT user EMPTY>
              <!ATTLIST user v CDATA #REQUIRED>
        <!ELEMENT group EMPTY>
              <!ATTLIST group v CDATA #REQUIRED>
```

<externaltask> Element

An external task runs external programs when it becomes active.

```
<!ELEMENT externaltask (description?, areavpath, successors,
                            command, timeout?, files?, activation?,
                            inactivate?, resets?, eastartop*,
                            eafinishop*, variables?)>
     <!ATTLIST externaltask owner CDATA #REQUIRED
                                name ID #REQUIRED
                                start (t|f) "f"
                                description CDATA #IMPLIED
                                lock (t|f) "f"
                                readonly (t|f) "f">
```

Attributes:

There are no attributes unique to external tasks.

See “Parameters Common to All Tasks” on page 177 for descriptions of the attributes shown in the preceding DTD excerpt.

Subelements:

See “<usertask> Element” on page 182 for a description of the `<successors>` subelement. See “Parameters Common to All Tasks” on page 177 for descriptions of the other subelements from the preceding DTD excerpt that are common to other tasks. Subelements that are unique to external tasks are as follows:

`<command>`

The `<command>` element specifies the full path of the program to be run on activation followed by any initial arguments. When the program is run by the workflow system, the following arguments are passed as separate arguments: the containing job's ID (in decimal), the ID of the task, and each file from the task's file list. On Solaris the program will be run as the owner of the task. On Windows NT it runs as the SYSTEM user.

Syntax for use of the `<command>` subelement is as follows:

```
<!ELEMENT command EMPTY>
    <!ATTLIST command v CDATA #REQUIRED>
```

When an external program finishes it must run the `iwcallback` program, passing the job and task IDs and a return code as arguments, to tell the server that it is finished. The server does not wait for an external task to finish. The server uses the return code passed to `iwcallback` to choose the set of successors to signal. If the return code is out of the range 0..*n*-1 (where *n* is the number of `<successorset>` elements), the last successor set is used.

<cgitask> Element

A CGI task behaves much like an external task. The only difference is that a CGI task does not run its `<command>` element (it relies on the user interface for that). A CGI task expects to have `iwcallback` called to notify it of program completion.

```
<!ELEMENT cgitask (description?, areavpath, successors, command,
                      timeout?, files?, activation?, inactivate?,
                      resets?, eastartop*, eafinishop*, variables?)>
    <!ATTLIST cgitask owner CDATA #REQUIRED
                          name ID #REQUIRED
                          start (t|f) "f"
                          description CDATA #IMPLIED
                          lock (t|f) "f"
                          immediate (t|f) "f"
                          readonly (t|f) "f">
```

Attributes:

There are no attributes unique to CGI tasks.

See “Parameters Common to All Tasks” on page 177 for descriptions of the attributes shown in the preceding DTD excerpt.

Subelements:

There are no subelements unique to CGI tasks.

See “<usertask> Element” on page 182 for a description of the `<successors>` subelement. See “Parameters Common to All Tasks” on page 177 for descriptions of the other subelements from the preceding DTD excerpt.

<submittask> Element

A submit task performs a submit operation on its contained files.

```
<!ELEMENT submittask (description?, areavpath, successorset,
                       timeout?, files?, activation?, inactivate?,
                       resets?, eastartop*, eafinishop*,
                       variables?)>
    <!ATTLIST submittask owner CDATA #REQUIRED
                           name ID #REQUIRED
                           start (t|f) "f"
                           skipconflicts (t|f) "f"
                           skiplocked (t|f) "f"
                           override (t|f) "f"
                           description CDATA #IMPLIED
                           lock (t|f) "f">
```

If the submit task succeeds, the successor tasks specified in the `<successorset>` element are signaled. If the submit task fails, the `submit` task goes into a special state that the user interface can detect. When the user interface has resolved any conflicts it retries the operation so that the job can continue. For the purposes of workflow, a submit task is considered successful even if some of its contained files were not submitted because of being up to date with the staging area.

Attributes:

See “Parameters Common to All Tasks” on page 177 for descriptions of the attributes shown in the preceding DTD excerpt that are common to other tasks. Attributes that are unique to submit tasks are as follows:

`skipconflicts`	Does not submit conflicting files.
`skiplocked`	Does not submit locked files.
`override`	Overwrites the staging area version of conflicting files.

Subelements:

There are no subelements unique to submit tasks.

See “<usertask> Element” on page 182 for a description of the `<successorset>` subelement. See “Parameters Common to All Tasks” on page 177 for descriptions of the other subelements from the preceding DTD excerpt.

<updatetask> Element

An update task does the equivalent of Get Latest (if the source is the staging area) or Copy To (if the source is another workarea or edition) on its contained files.

```
<!ELEMENT updatetask (description?, areavpath, successorset,
                        srcareavpath, timeout?, files?, activation?,
                        inactivate?, resets?, eastartop*,
                        eafinishop*, variables?)>
    <!ATTLIST updatetask owner CDATA #REQUIRED
                            name ID #REQUIRED
                            start (t|f) "f"
                            delete (t|f) "t"
                            overwritemod (t|f) "f"
                            description CDATA #IMPLIED
                            lock (t|f) "f">
```

If the update task fails because of conflicts, it goes into a state like that of a failed submit task. The user interface is responsible for resolving conflicts and retrying the update task.

Attributes:

See “Parameters Common to All Tasks” on page 177 for descriptions of the attributes shown in the preceding DTD excerpt that are common to other tasks. Attributes that are unique to update tasks are as follows:

`delete`	Propagates deleted files to the destination area.
`overwritemod`	Overwrites conflicting versions of files in the destination area.

Subelements:

See “<usertask> Element” on page 182 for a description of the `<successorset>` subelement. See “Parameters Common to All Tasks” on page 177 for descriptions of the other subelements from the preceding DTD excerpt that are common to other tasks.

Subelements that are unique to update tasks are as follows:

`<srcareavpath>`

The area from which files are copied.

<endtask> Element

An end task is a marker for the end of a job. When an end task becomes active, its associated job is terminated and all locks held in the job are released.

```
<!ELEMENT endtask (activation?, eastartop*, eafinishop*)>
     <!ATTLIST endtask name ID #REQUIRED
                          description CDATA #IMPLIED>
```

Attributes:

There are no attributes unique to end tasks.

See “Parameters Common to All Tasks” on page 177 for descriptions of the attributes from the preceding DTD excerpt.

Subelements:

There are no subelements unique to end tasks.

See “Parameters Common to All Tasks” on page 177 for descriptions of the subelements from the preceding DTD excerpt.

<dummytask> Element

A `<dummytask>` element is a task that waits for its mandatory timeout to expire. If "+000000" is specified as a timeout value, `<dummytask>` becomes a spacer task. Dummy tasks let a workflow designer create a time interval unrelated to any actual job activity. A dummy task does not have an owner or areavpath.

```
<!ELEMENT dummytask (description?, timeout, files?, activation?,
                         inactivate?, resets?, eastartop*,
                         eafinishop*, variables?)>
     <!ATTLIST dummytask name ID #REQUIRED
                            start (t|f) "f"
                            description CDATA #IMPLIED>
```

Attributes:

There are no attributes unique to dummy tasks.

See "Parameters Common to All Tasks" on page 177 for descriptions of the attributes from the preceding DTD excerpt.

Subelements:

There are no subelements unique to dummy tasks.

See "Parameters Common to All Tasks" on page 177 for descriptions of the subelements from the preceding DTD excerpt.

<locktask> Element

A `<locktask>` element is a task that attempts to acquire locks on the files it owns. If it succeeds, it transitions to the successors specified in its `success` element. If it fails, it transitions to the successors specified in its `failure` element. This provides users with a way of backing out of a job or choosing an alternate path in a job that cannot acquire its locks.

```
<!ELEMENT locktask (description?, areavpath, success, failure,
                        files?, activation?, inactivate?, resets?,
                        eastartop*, eafinishop*, variables?)>
    <!ATTLIST locktask owner CDATA #REQUIRED
                           name ID #REQUIRED
                           start (t|f) "f"
                           description CDATA #IMPLIED>
    <!-- Locking is implied -->
    <!ELEMENT success (succ+)>
    <!ELEMENT failure (succ+)>
```

Attributes:

There are no attributes unique to lock tasks.

See "Parameters Common to All Tasks" on page 177 for descriptions of the attributes from the preceding DTD excerpt.

Subelements:

See "Parameters Common to All Tasks" on page 177 for descriptions of the subelements from the preceding DTD excerpt that are common other tasks.

Subelements that are unique to lock tasks are as follows:

`<success>`

Names the successor tasks that become active when the lock task succeeds.

`<failure>`

Names the successor tasks that become active when the lock task fails.

Perl Modules

These Perl modules are provided as reference for workflow template developers. Refer to the Perl modules for the latest documentation, or see `iw-home/iw-perl/bin/perldoc` *`module`*.

TeamSite::WFsystem

Synopsis

Utilities for accessing the TeamSite workflow engine. This provides access to functions for querying the entire workflow system.

```
use TeamSite::WFsystem;
$system = new TeamSite::WFsystem();
```

Functions

`new()`	Creates a new `WFsystem` object. This only works on the local TeamSite server.
`GetWorkflows()`	Returns an array of `WFworkflow` objects corresponding to all jobs in the system.
`GetActiveWorkflows()`	Returns an array of `WFworkflow` objects corresponding to all active jobs in the system.
`GetTasks()`	Returns an array of `WFworkflow` objects corresponding to all tasks in the system.
`GetActiveTasks()`	Returns an array of `WFtask` objects corresponding to all active tasks in the system.
`CreateWorkflow($spec) , CreateWorkflow($spec,$tmpfile)`	Creates a workflow instance from `$spec` (a workflow specification in the form of a string). Returns a `TeamSite::WFworkflow`. If the `$tmpfile` arg is set, `tmpfile` is used instead of `stdin`. Using this option in the context of HTTP may result in timing security issues.
`Refresh()`	Call after changes have been made.

Examples

```
$system = new TeamSite::WFsystem();
$wfs = $system->GetWorkflows();
$wfs = $system->GetActiveWorkflows();
```

`$wfs` is a reference to an array containing references to `TeamSite::WFworkflow` objects.

```
$tasks = $system->GetTasks();
$tasks = $system->GetActiveTasks();
```

`$tasks` is a reference to an array containing `TeamSite::WFtask` references.

```
my $wf_system = TeamSite::WFsystem->new();
my $wf        = $wf_system->CreateWorkflow($spec);
if ((!defined $wf) || !$wf->IsValid() ||  $wf->GetError())
{
 ... handle error...
}
```

TeamSite::WFworkflow

Synopsis

Utilities for using the TeamSite workflow engine. This supplies functions for manipulating and querying workflows.
`$workflow = new TeamSite::WFworkflow($id)`

Functions

`new($id)`	Creates a new `WFworkflow` object.
`GetId()`	Fetches the workflow ID.
`GetError()`	Fetches the last error message (if any).
`SetError($error_string)`	Sets the error message to `$error_string` and returns the previous error message (if any).
`IsValid()`	Determines whether this a valid workflow object.
`GetTasks()`	Gets the tasks owned by this workflow.
`GetOwner()`	Returns owner of workflow.

`GetCreator()`	Returns the creator of this workflow.
`GetName()`	Returns the name of the workflow.
`GetDescription()`	Returns the description for this workflow.
`Invoke()`	Starts this workflow running. Returns a `TeamSite::WFtask` object. If the returned object is valid, then a CGI task that wishes to be run.
`GetVariable($name)`	Gets the value of a workflow variable.
`SetVariable($name, $value)`	Sets the value for a workflow variable. Returns exit status of underlying CLT (non-zero indicates an error occurred).
`CreateVariable($name, $value)`	Creates a workflow variable. If the variable already exists, this fails.
`DeleteVariable($name)`	Deletes a workflow variable.
`Refresh()`	Call when workflow object has been modified.

Examples

```
$workflow = new TeamSite::WFworkflow($id);
$tasks = $workflow->GetTasks();
```

`$tasks` is a reference to a list containing `TeamSite::WFtask` objects.

TeamSite::WFtask

Synopsis

Utilities for using the TeamSite workflow engine. This supplies functions for manipulating and querying tasks.

```
$task = new TeamSite::WFtask($id);
```

Functions

`new($id)`	Creates a new `WFtask` object.
`GetId()`	Returns the task ID.
`GetType()`	Returns the task type.
`GetOwner()`	Gets the owner of this task.
`GetDescription()`	Returns the description for this task.
`GetWorkflowId()`	Returns the ID of the job that owns this task.
`AddFile($path, $comment)`	Adds a file with comment to a task.
`SetComment($comment)`	Sets comment on task.
`($success, $immediatetask) SelectTransition($which, $comment)`	Selects a transition for this task. `$success` is a boolean and `$immediatetask` is a possibly invalid `TeamSite::WFtask` to run.
`($success, $immediatetask) = CallBack($retcode, $comment)`	Callback from a CGI task or external task. `$immediatetask` is a possibly invalid `TeamSite::WFtask` to run.
`GetCommand()`	Gets the command string for an external task.
`Refresh()`	Call when the server side object has been changed.
`IsValid()`	Returns true if this is a valid task.
`GetSubmitEvents()`	Returns a (possibly empty) array of `SubmitEvent` objids (as strings). It returns an array because there may have been conflicts or other problems which could produce multiple events.

`GetUpdateEvents()`	Returns a (possibly empty) array of `UpdateEvent objids` (as strings). It returns an array because there may have been conflicts or other problems that could produce multiple events.
`GetFiles()`	Returns a (possibly empty) array of file names.
`GetArea()`	Gets the area for the task, such as `/default/main/dev/WORKAREA/andre`
`GetError()`	Retrieves the last error message (if any).
`SetError($error_string)`	Sets the error message to `$error_string` and returns the previous error message (if any).

Example

```
$task = new TeamSite::WFtask($id);
```

Sample Job Specification File

The following job specification file could be created by direct editing (see page 173) or by configuring a workflow template file to generate it based on data provided by an end-user. This file defines a workflow for this sequence of events:

1. A worker named Mark generates a set of documentation about a new product called B4000.
2. A worker named Bill then receives this documentation and prepares it for the web.
3. Bill's manager and the legal department review Bill's and Mark's efforts.
4. Material is submitted and deployed on the live web server.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE workflow SYSTEM "iwwf.dtd">

<!-- Sample workflow for B4000. -->

<workflow name="B4000" owner="BillsManager"
    description="Standard workflow for new product information.">
 <usertask name="MarkWork" owner="Mark"
    description="Write copy for B4000" start="t">
    <areavpath v="/default/main/WORKAREA/Mark"/>
    <successors>
       <successorset description="Done">
          <succ v="MarkToBill"/>
       </successorset>
    </successors>
    <activation>
       <or>
          <pred v="BillToMark"/>
          <pred v="ReviewToMark"/>
       </or>
    </activation>
 </usertask>

 <updatetask name="MarkToBill" owner="Bill"
       description="Update Bill&apos;s Workarea">
    <areavpath v="/default/main/WORKAREA/Bill"/>
    <successorset>
       <succ v="BillWork"/>
    </successorset>
    <srcareavpath v="/default/main/WORKAREA/Mark"/>
    <activation>
       <pred v="MarkWork"/>
    </activation>
 </updatetask>
```

```
<usertask name="BillWork" owner="Bill"
      description="Webify this doc.">
   <areavpath v="/default/main/WORKAREA/Bill"/>
   <successors>
      <successorset description="Done">
         <succ v="BillToReview"/>
      </successorset>
      <successorset description="Send back to Mark">
         <succ v="BillToMark"/>
      </successorset>
   </successors>
   <activation>
      <or>
         <pred v="MarkToBill"/>
         <pred v="ReviewToBill"/>
      </or>
   </activation>
</usertask>

<updatetask name="BillToReview" owner="Manager"
   description="Update the Review area from Bill&apos;s
                  Workarea.">
   <areavpath v="/default/main/WORKAREA/Review"/>
   <successorset>
      <succ v="LegalReview"/>
      <succ v="ManagerReview"/>
   </successorset>
   <srcareavpath v="/default/main/WORKAREA/Bill"/>
   <activation>
      <pred v="BillWork"/>
   </activation>
</updatetask>
```

```
<usertask name="LegalReview" owner="Legal"
      description="Limit exposure." readonly="t">
   <areavpath v="/default/main/WORKAREA/Review"/>
   <successors>
      <successorset description="Okay">
         <succ v="Submit"/>
      </successorset>
      <successorset description="Legal problem">
         <succ v="ReviewToMark"/>
      </successorset>
   </successors>
   <activation>
      <pred v="BillToReview"/>
   </activation>
</usertask>

<usertask name="ManagerReview" owner="Manager"
      description="Final Approval" readonly="t">
   <areavpath v="/default/main/WORKAREA/Review"/>
   <successors>
      <successorset description="Okay">
         <succ v="Submit"/>
      </successorset>
      <successorset description="Send back to Mark">
         <succ v="ReviewToMark"/>
      </successorset>
      <successorset description="Send back to Bill">
         <succ v="ReviewToMark"/>
      </successorset>
   </successors>
   <activation>
      <pred v="BillToReview"/>
   </activation>
</usertask>
```

```
<submittask name="Submit" owner="Manager"
      description="Final submission.">
   <areavpath v="/default/main/WORKAREA/Review"/>
   <successorset>
      <succ v="Deploy"/>
   </successorset>
   <activation>
      <and>
         <pred v="LegalReview"/>
         <pred v="ManagerReview"/>
      </and>
   </activation>
</submittask>

<externaltask name="Deploy" owner="Manager"
      description="Deploy to live server.">
   <areavpath v="/default/main/STAGING"/>
   <successors>
      <successorset description="Successful Deployment">
         <succ v="End"/>
      </successorset>
      <successorset description="Deployment failed">
         <succ v="End"/>
      </successorset>
   </successors>
   <command v="/scriptorium/do_deploy.pl"/>
   <activation>
      <pred v="Submit"/>
   </activation>
</externaltask>

<endtask name="End">
   <activation>
      <pred v="Deploy"/>
   </activation>
</endtask>

<!-- Various send back updates -->
```

```
  <updatetask name="ReviewToBill" owner="Bill"
        description="Update Bill&apos;s workarea form the Review
                       workarea.">
     <areavpath v="/default/main/WORKAREA/Bill"/>
     <successorset>
        <succ v="BillWork"/>
     </successorset>
     <srcareavpath v="/default/main/WORKAREA/Review"/>
     <activation>
        <pred v="ManagerReview"/>
     </activation>
  </updatetask>

  <updatetask name="BillToMark" owner="Mark"
        description="Update Mark&apos;s workarea from Bill&apos;s">
     <areavpath v="/default/main/WORKAREA/Mark"/>
     <successorset>
        <succ v="MarkWork"/>
     </successorset>
     <srcareavpath v="/default/main/WORKAREA/Bill"/>
     <activation>
        <pred v="BillWork"/>
     </activation>
  </updatetask>

  <updatetask name="ReviewToMark" owner="Mark"
        description="Update Mark&apos;s workarea from Review">
     <areavpath v="/default/main/WORKAREA/Mark"/>
     <successorset>
        <succ v="MarkWork"/>
     </successorset>
     <srcareavpath v="/default/main/WORKAREA/Review"/>
     <activation>
        <or>
           <pred v="ManagerReview"/>
           <pred v="LegalReview"/>
        </or>
     </activation>
  </updatetask>

</workflow>
```

Appendix A

The iwsend_mail.ipl Script

The Perl script `iwsend_mail.ipl` was specifically designed for use within TeamSite workflows to simplify the creation of external task scripts for email notification. The script is installed by default in the `/iw-home/bin` directory.

What's New In iwsend_mail.ipl for TeamSite 5.5.2?

The following features were added to the version of this script that shipped with versions of TeamSite prior to Release 5.0:

- Mapping of TeamSite user names to their corresponding email addresses
- Incorporation of all workflow comments and transition comments into the email
- Enabling URL references to be used on the file list included in the email
- Command line arguments to specify multiple To and Cc recipients, the From field, the subject line, and main body of the message
- Functional API call `TeamSite::WFtask::GetComments()` replaces the `get_transition_comments` subroutine

The sections that follow describe these features in a detail (refer to the OpenAPI documentation for information on the `TeamSite::WFtask::GetComments()` call).

Configuring iw.cfg with Site-specific Information

Complete the following procedure to modify your `iw.cfg` file to include certain required and optional parameters for the `iwsend_mail.ipl` script to work.

1. Open the `iw.cfg` file.

   By default, it is located in either:

   - `/etc` (Solaris servers)
   - `c:\Program Files\Interwoven\TeamSite\etc` (Windows servers)

2. Modify the `[iwsend_mail]` section to include the following lines:

```
[iwsend_mail]
maildomain=interwoven.com
mailserver=mail1.interwoven.com
use_mapping_file=true
email_mapping_file=c:/iw-home/local/config/wft/email_map.cfg
debug_output=c:/tmp/iwsend_mail.log
```

   Notes:

   - `maildomain`: Required entry that must be set to the email domain used for email addresses that are not otherwise qualified with a domain address.
   - `mailserver`: Required entry that specifies the mail server used for SMTP.
   - `use_mapping_file`: Optional entry (default=false). If this is set to true, all email addresses are matched against a specified mapping file to see if they need to be altered from their present settings.
   - `email_mapping_file`: Required entry if `use_mapping_file=true`. This specifies the full path to the email mapping file (more details below).
   - `debug_output`: Optional entry. If this option is set, debugging information is sent to the file specified.

3. Save and close the file.

Determining Email Addresses

The value of `use_mapping_file` and `email_mapping_file` determines which of two areas of the `convert_email()` section are performed.

- When `use_mapping_file` is set to false (the default), `iwsend_mail.ipl` uses the TeamSite username that has been passed to the script for the email address. If this username contains a Windows NT domain before the name, it is removed and only the username is used for the email address. If the value passed contains an "@" symbol it is not changed. Otherwise, the maildomain value is appended.
- When `use_mapping_file` value is changed to true, the script uses a flat file to map the TeamSite username to a corresponding email address. The TeamSite administrator must maintain the flat file whenever new users are added. The script functionality should be described before the format of the file. The script functionality:
  - Creates a default email address by using the values of the specified recipients as passed into the script.
  - Opens the email mapping flat file and parses the contents searching for a match for the specific recipient. If a match is found, `$email_address` is set to the corresponding value. If no match is found, the recipient value is left unchanged.
  - In both cases `@maildomain` is appended if needed.

  The email mapping flat file must use the following format:

  - Its location is specified by the `email_mapping_file` configuration option, for example:

    `IWHOME/local/config/email_map.cfg.`

  - It contains a list of names followed by a colon, and the email address, for example:

    ```
    tsuser1: jsantoro
    tsuser2: someone@some_domain
    tsuser3: bgunn@interwoven.com
    ```

Command Line Arguments

The `iwsend_mail.ipl` script includes command line functionality for greater flexibility when sending workflow related notifications. This functionality includes multiple recipients, mail sender identification, user-configurable subject lines and message bodies.

The usage for `iwsend_mail.ipl` is:

```
iwperl iwsend_mail.ipl <userid> <jobid> <taskid>
```

The command in the workflow would look like:

```
<command v="d:/iw-home/iw-perl/bin/iwperl d:/iw-home/bin/
iwsend_mail.ipl userid"/>
```

The workflow engine automatically supplies the `jobid` and `taskid` (and area), but they must be added manually if you are going to run the script from the command line.

The following sections describe these arguments individually and then show an example where they are combined to generate a complete notification.

Multiple Email Recipients

You can specify multiple email recipients on the command line for both **To:** and **Cc:** fields. This can be accomplished in either of two ways (or using a combination of both):

- `-t recipient1 –t recipient2 […]`
- `-t "recipient1,recipient2[,…]" -c recipient3 […]`

The `-t` flag corresponds to the **To:** field and the `-c` flag corresponds to the **CC:** field.

Note: There must be at least one `-t` specification.

Mail Sender

You can specify who the sender of the message is supposed to be by using the `-f sender` command. The sender address is validated the same way as each recipient's name. The default value is the task's owner.

Subject Line

The default subject line is `TeamSite Task Notification`. You can replace the default message by using the `-s` argument and entering the desired subject enclosed in double quotes. For example:

```
-s “Top Priority Task Notification”
```

Message Body

By default, the message body contains the following parts:

- Summary information (Job Id, Areavpath, Job Name, Job Description)
- Message (the default is: `A task in job` *`JobId`* `has been assigned to you.`)
- List of comments associated with the task (if any)
- List of files associated with the task (if any)

You can replace the message portion with your own text by using either:

- `-m “my one line message text”`
- `-b “`*`/path_to_file_containing`*`/message.txt”`

If both options are specified, the `-m` option is ignored.

Example

The following example assumes a Solaris system (to simplify the command line). If you make the modifications as specified, the resulting email will look similar to the sample email message on page 207.

- In the `iw.cfg` file:

```
[iwsend_mail]
maildomain=my_company.com
mailserver=smtp.my_company.com
use_mapping_file=true
email_mapping_file=/usr/iw-home/local/config/email_map.cfg
```

- In the `email_map.cfg` file:

```
tsuser1: jsantoro
tsuser2: someone@some_domain
tsuser3: vvenkata
```

- In instantiated workflow, a command specification in the workflow like the following (note: this is one long line that has wrapped):

```
<command v='/usr/iw-home/bin/iwsend_mail.ipl -t "tsuser1, tsuser2" -
c chico -f Harpo@Marx-Brothers.com -m "This Space Available for
Advertising" -s "Sample Subject Line"'/>
```

Sample Email Message

```
Subject: Sample Subject Line
To: jsantoro@my_company.com, someone@some_domain
From: Harpo@Marx-Brothers.com
Cc: chico@my_company.com
X-Mailer: Mail::Mailer[v1.18] Net::SMTP[v2.15]
Date: Thu, 22 Feb 2001 13:04:20 -0500 (EST)

======================================================================

Job: 274972
Area: \default\main\devnet\WORKAREA\shared
Name: fmailTestWorkflow
Description: Demonstration of new iwsend_mail.ipl script

----------------------------------------------------------------------

Date: Thu Feb 2 04:20:00 2001  Task: 274973  User: ZASTOLLERLNS\ghoti

> Transitioning from first user task to first externaltask using
> the new iwsend_mail.ipl

What do you think?

----------------------------------------------------------------------

============================ File list ===============================

> move_files.ipl
> msdw_deploy.ipl
> msdw_rmreplicant.ipl

----------------------------------------------------------------------
```

Appendix B

Creating a Nested Job

TeamSite 5.5.2 enables workflow developers to create *nested workflow*—workflow that is contained within other jobs or tasks. The implementation of nested workflow is similar to external and CGI tasks where the activation of workflow tasks is either automatically or manually instantiated causing an association of a new job with the workflow task. The nesting process creates a parent/child relationship with the task as the parent and the job as the child.

The relationship between a workflow task and its nested workflow includes:

- the ability to pass variables and file lists from the parent task to the child job
- the ability for nested jobs to pass some or all variables and file lists to the parent job upon the child job's completion
- the lifetime of a nested job is dependent upon its parent task's workflow lifetime—it should not be removed from the backing store until its parent task is deleted

Workflow tasks can either be specified with a path to a *job specification file* or to a *workflow template file* (.wft). In the case of a job specification file, upon activation of the workflow task, TeamSite compiles and instantiates a new job using the specification file. In the case of a workflow template file, the task owner must manually start the task using the New Job window to input job variables and subsequently initiate the job.

Creating Jobs with Nested Workflow

Complete the following procedure to create a job with a nested workflow task. This procedure assumes that the `author_assignment_with_nested_job.wft` file installed with WorkflowBuilderis specified in your `available_templates.cfg` file. If it is not, you can locate the sample workflow file in `iw-home/local/config/wft/examples` and add it to your `available_templates.cfg` file.

The `author_assignment_with_nested_job.wft` file defines a job that contains two tasks, the second of which does not begin until the first has been approved by an editor.

1. Log into TeamSite using the Editor, Master, or Administrator role.
2. Select **File** > **New Job** from the drop-down menu.

   The New Job window is displayed.
3. Complete the following steps in the New Job window:

   a. Select the **Author Assignment with Optional Nested Job** template.

   b. Type a description of the new job, for example: **`Test of Nested Workflow`**.

   c. Click **New Job**.

   This executes the `iwwft_instantiator.cgi` to instantiate the job. The New Job Template window is displayed with the description you entered in step b in the **Job Description** field.
4. Complete the following steps in the New Job Template window:

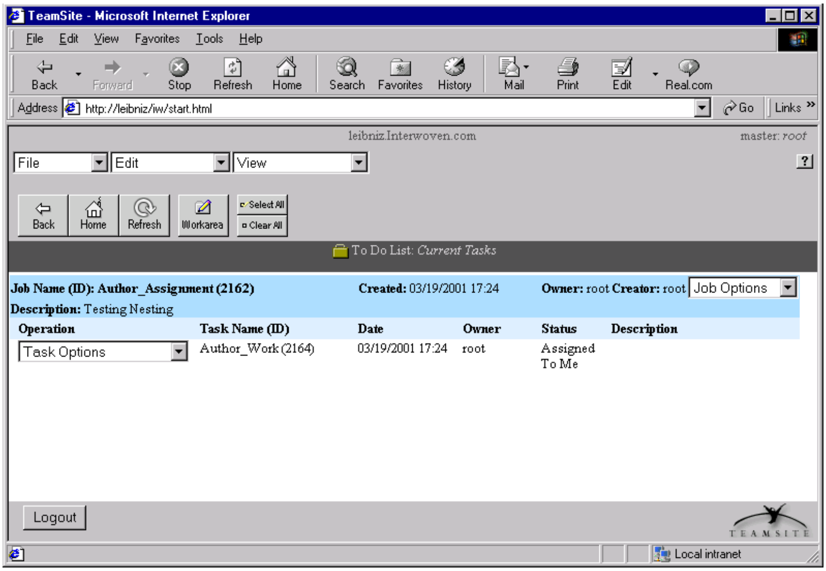
   a. Select an Author from the drop-down menu (to make this demonstration easier, select the same user as you are currently logged in as).

   b. Select a Branch from the drop-down menu.

   c. Type the name of a workarea in the Enter Workarea field.

   d. Click **Run Job**.

5. In the main TeamSite window, click **To Do** to display the assignment of the job you just created.



Note the following in the graphic:

– The Owner and Creator are both root.

– This screen does not make any mention of the nesting—it is invisible to the person to which the task is assigned.

Index

Symbols

A

B

C

D

E

F

W

X

Z