



I N T E R W O V E N

**Interwoven TeamXpress™
for Multiplatforms V1.1,
WebSphere™ Edition**

Templating and Deployment Guide

© 2001 Interwoven, Inc. All rights reserved.

No part of this publication (hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Interwoven. Information in this manual is furnished under license by Interwoven, Inc. and may only be used in accordance with the terms of the license agreement. If this software or documentation directs you to copy materials, you must first have permission from the copyright owner of the materials to avoid violating the law, which could result in damages or other remedies.

Interwoven, TeamSite, OpenDeploy, and the logo are registered trademarks of Interwoven, Inc., which may be registered in certain jurisdictions. TeamXpress, SmartContext, DataDeploy, Content Express, the tagline and service mark are trademarks of Interwoven, Inc., which may be registered in certain jurisdictions. All other trademarks are owned by their respective owners.



I N T E R W O V E N

Interwoven, Inc.

1195 West Fremont Ave.

Sunnyvale, CA 94087

<http://www.interwoven.com>

Printed in the United States of America

Release 1.1

Part # 40-00-40-45-04-110-310

Table of Contents

About This Book 5

- Notation Conventions 6
- Windows Path Name Conventions 7
- Support Information 7

Section 1: TeamXpress Templating

Chapter 1: Installing TeamXpress Templating 11

- Hardware Requirements 11
- Operating System Requirements 11
- Installing on Solaris 11
- Installing on Windows NT/2000 13
- Installing on Client Machines 13
- Next Step 14

Chapter 2: Initial Configuration 15

- Configuration Overview 15
- Configuring the Example Templating Environment 29
- Proxy Server Configuration 34
- Starting TeamXpress Templating 35

Chapter 3: Setting Up Data Capture Templates 37

- Data Capture Template Overview 38
- Example Data Capture Templates 39
- Data Capture Example 1 39
- Data Capture Example 2 57
- Data Capture Template DTD 63

Chapter 4: Setting Up Presentation Templates 69

- Creating Presentation Templates 69
- Custom XML Tags 82
- Writing Your Own Tags 127

Chapter 5: Mapping Users, Templates, and Content Records 129

- templating.cfg Overview 129

Example templating.cfg File 130
templating.cfg DTD 136

Chapter 6: Integrating Templating, DataDeploy, and Workflow 139

Integration Overview 139
Integration Steps 140

Section 2: DataDeploy Administration

Chapter 7: Overview and Installation 145

Overview 145
Client/Server Setup Options 146
Installing DataDeploy 148

Chapter 8: Deployment Concepts 151

Ways to Invoke Deployment 151
Configuration Files 152
Data Organization Within DataDeploy 154
Deployment Scenarios 158

Chapter 9: Configuration File Details and Examples 171

Required Elements 171
Parameter Substitutions 175
Sample TeamXpress-to-Database Configuration File 175
Sample TeamXpress-to-XML Configuration File 196
Sample Database-to-Database Configuration File 197
Sample Database-to-XML Configuration File 198
Sample XML-to-Database Configuration File 200
Sample XML-to-XML Configuration File 202
Starting-State Base Table Configuration File 204
Event 1 Configuration File 205
Event 2 Configuration File 206

Chapter 10: Invoking DataDeploy 207

iwdd.ipl Command 207
Running DataDeploy as a Service 210

Chapter 11: Synchronizing OpenDeploy and Data Deploy 211

- Overview 211
- Software Requirements 213
- Program and Configuration Files 213
- Synchronized Deployment Process 214
- Configuring OpenDeploy 217
- Configuring DataDeploy 222
- Invoking Synchronized Deployment 227

Section 3: OpenDeploy Administration

Chapter 12: Installing OpenDeploy 231

- UNIX 232
- Windows NT/2000 235

Chapter 13: Syntax and Options 239

- iwdeploy Syntax 239
- Options 242

Chapter 14: Configuration Files 251

- OpenDeploy Server Configuration Files 251
- OpenDeploy Client Configuration Files 253
- Scope of Configuration File Options 255
- The Authorization Configuration File 258

Chapter 15: Configuration File Options 261

- OpenDeploy Client Options 261
- OpenDeploy Server Options 289

Chapter 16: Advanced Features 301

- Authentication by IP Address 301
- Encryption 305
- Deploy and Run 313

Chapter 17: Deployment Scenarios 325

- Forward Deployment to a Single Server 326
- Forward Deployment to Multiple Servers 329
- Forward Deployment of Different Directories to Different Servers 333

Reverse Deployment 337
Reverting Websites to Previous Versions 346
Deploying Through Firewalls 349

Section 4: Appendices

Appendix A: Creating Data Capture Templates from DTDs 353

Running the CLT on the DTD File 354
The symbol-table.cfg File 354
The datacapture.cfg File 358
Diagram Key 360
Unsupported DTD Features 360
Symbol Table DTD Used for Conversions 361

Appendix B: Using Command-Line Tools 367

Appendix C: DataDeploy Database Auto-Synchronization 383

Overview 383
Software Requirements 383
DAS Program and Configuration Files 384
Configuring DAS 385
Using DAS 392
TeamXpress Event Triggers 396
Logging DAS Activities 398
Disabling DAS 399
iwsyncdb.ipl Usage 399

Appendix D: DataDeploy Database Server Configuration 403

Overview 403
IBM DB2 403
Sybase ASE 404
Informix 405

Appendix E: DataDeploy Querying Tables 407

Appendix F: OpenDeploy Client and Server Configuration File Options 409

Index 413

About This Book

The *TeamXpress Templating and Deployment Guide* contains information on how to:

- install and configure TeamXpress Templating
- develop presentation templates and data capture templates
- install, configure, and use DataDeploy with TeamXpress OpenDeploy
- install and configure OpenDeploy

It is primarily intended for TeamXpress developers and for web server administrators and system administrators. Many of the operations described in this manual require root (Solaris) or Administrator (Windows NT® or Windows 2000®) access to the TeamXpress server. If you do not have root or Administrator access to the TeamXpress server, consult your system administrator.

Windows NT/2000: Users should be familiar with either IIS or Netscape web servers, and with basic Windows NT/2000 operations such as adding users and modifying ACLs (Access Control Lists).

Solaris: Users of this manual should be familiar with basic UNIX commands and be able to use an editor such as emacs or vi.

It is also very helpful to be familiar with regular expression syntax. If you are not familiar with regular expressions, it is recommended that you consult a reference manual such as *Mastering Regular Expressions*, by Jeffrey Friedl.

Notation Conventions

This manual uses the following notation conventions:

Convention	Definition and Usage
Bold	Text that appears in a GUI element (e.g., a menu item, button, or element of a dialog box) and command names are shown in bold. For example: Click Edit File in the Button Bar.
<i>Italic</i>	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 <code>iwextattr</code> command-line tool allows you to set and look up extended attributes on a file.
<i>Monospaced italic</i>	Monospaced italics are used for command-line variables.
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: <code>% iwextattr -s project=proj1 //IWSERVER/default/main/dev/WORKAREA/andre/products/index.html</code>
<i>Monospaced bold italic</i>	Monospaced bold italic text is used to indicate a variable in user input. For example: <code>% iwextattr -s project=<i>projectname</i> <i>workareavpath</i></code> means that you must insert the values of <i>projectname</i> and <i>workareavpath</i> 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.

Windows Path Name Conventions

In most cases, you can specify path names using standard Windows NT/2000 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/2000 naming conventions, enter the path name again using MS-DOS conventions. For example, instead of:

```
>C:\iw-home\Program Files\Interwoven
```

you can try:

```
>C:\iw-home\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.

Support Information

For support information concerning IBM TeamXpress, refer to the following URL:
<http://www-4.ibm.com/software/web servers/teamxpress/support.html>.



Section 1: TeamXpress Templating

-
- Installing TeamXpress Templating
 - Initial Configuration
 - Setting Up Data Capture Templates
 - Setting Up Presentation Templates
 - Mapping Users, Templates, and Content Records
 - Integrating Templating, DataDeploy, and Workflow



Installing TeamXpress Templating

TeamXpress 1.0 must be installed on your system before you can install TeamXpress Templating 1.0. If it is not, see the *TeamXpress Administration Guide* for installation instructions. Return to this chapter after TeamXpress is installed.

Hardware Requirements

TeamXpress Templating should be installed on a dual CPU server if you plan to enable data content record searches. See the *TeamXpress Administration Guide* for general information on hardware requirements. On client machines, at least 20 MB of hard disk space is required.

Operating System Requirements

TeamXpress Templating is supported by all of the operating systems that support TeamXpress. See the *TeamXpress Administration Guide* for information about supported operating systems.

Installing on Solaris

The TeamXpress templating package for Solaris is available in two forms: a compressed `pkgadd` package stream file or a package directory. If you have downloaded the Templating package, it will be in the compressed package stream form. If you are installing from the CD-ROM, it will be in the package directory form.

To install the package stream package, perform these steps:

1. Log in as **root**.
2. If a previous version of TeamXpress Templating was installed, issue the command:

```
# pkgrm IW0Vtst
```



3. Unzip and transfer the package stream package into a temporary location by issuing the following command (on one line), where *temp_dir* is a temporary directory with at least 128 megabytes of free space:

```
# gunzip < tst.4.5.0.Buildxxxx.pkg.gz | pkgtrans /dev/fd/0 temp_dir  
IWOVtst
```

4. Install TeamXpress Templating by issuing the following command:

```
# pkgadd -d temp_dir IWOVtst
```

5. Remove the temporary directory:

```
# rm -r temp_dir/IWOVtst
```

To install the package directory form, perform these steps:

1. Log in as **root**.
2. Change to the directory containing the `IWOVtst` directory. If you are installing from CD-ROM, this would be:

```
# cd /cdrom
```

3. Install TeamXpress Templating by issuing the following command:

```
# pkgadd -d . IWOVtst
```

Once TeamXpress Templating is installed, you must restart the `iwproxy` daemon:

1. Log in as **root**.
2. Issue the following commands:

```
# /etc/init.d/iw.server stop  
# /etc/init.d/iw.server start
```

See the *TeamXpress Administration Guide* for more information about restarting the proxy server.

Installing on Windows NT/2000

Perform the following steps to install TeamXpress Templating on TeamXpress running on a Windows NT/2000 system:

1. Log into Windows NT/2000 with Administrator permissions.
2. Insert the TeamXpress Templating CD into the CD drive. Navigate to the top-level directory and double click the `templating.exe` icon. The Interwoven TeamXpress Templating Setup screen appears.
3. Click **Next**. A dialog box appears, prompting for the destination of the TeamXpress Templating administrative files. It is recommended that you select the default location. If you specify a new location, it must *not* be the `iw-home` directory.
4. Click **Next**. File names are displayed while the TeamXpress Templating administrative files are loaded.
5. Click **OK**. The TeamXpress Templating directory structure shown on page 21 is installed in `iw-home`.
6. Restart the proxy server:
 - Select **Settings > Control Panel** from the **Start** menu.
 - Open the **Services** control panel.
 - Select **Interwoven Proxy** from the list of services.
 - Click **Stop** and wait for service to terminate.
 - Click **Start**. See Chapter 7 in the *TeamXpress Administration Guide* for more information about restarting the proxy server.

Installing on Client Machines

After TeamXpress Templating is installed and configured on the server, it is available for content contributors on client machines. When content contributors select **File > New Data Record**, they are prompted to install the client-side software. Refer to the *TeamXpress User's Guide* for the procedures.

Next Step

After you install TeamXpress Templating, you are ready to configure the example templating environment as described in the Chapter 2, “Initial Configuration.”

Initial Configuration

After TeamXpress Templating is installed on your system, you should perform the initial configuration described in this chapter. This initial configuration provides a fully functional example TeamXpress Templating environment to verify that the TeamXpress Templating installation was successful. You can also use the example templating environment to become familiar with TeamXpress Templating features. After you are familiar with the example templating environment, you can customize it to create your own site-specific templating environment as described later in this manual. The configuration activities described in this chapter should be performed by a system administrator.

This chapter begins with an overview of TeamXpress Templating configuration, followed by the initial setup activities that will create the example templating environment.

Configuration Overview

TeamXpress Templating provides a highly configurable way to capture, edit, and store data input from content contributors; define the appearance of displayed data; and integrate captured data with other products such as TeamXpress Workflow and DataDeploy. The TeamXpress Templating mechanism for capturing data content from content contributors is separate from the mechanism for defining the appearance of the content when it is displayed. This architecture allows for unlimited reuse of data after the data is captured and stored. It also lets you define different appearances and behaviors for the same data content based on how, when, where, or to whom the data is displayed. You can also use Perl code to generate content from other sources such as relational databases.

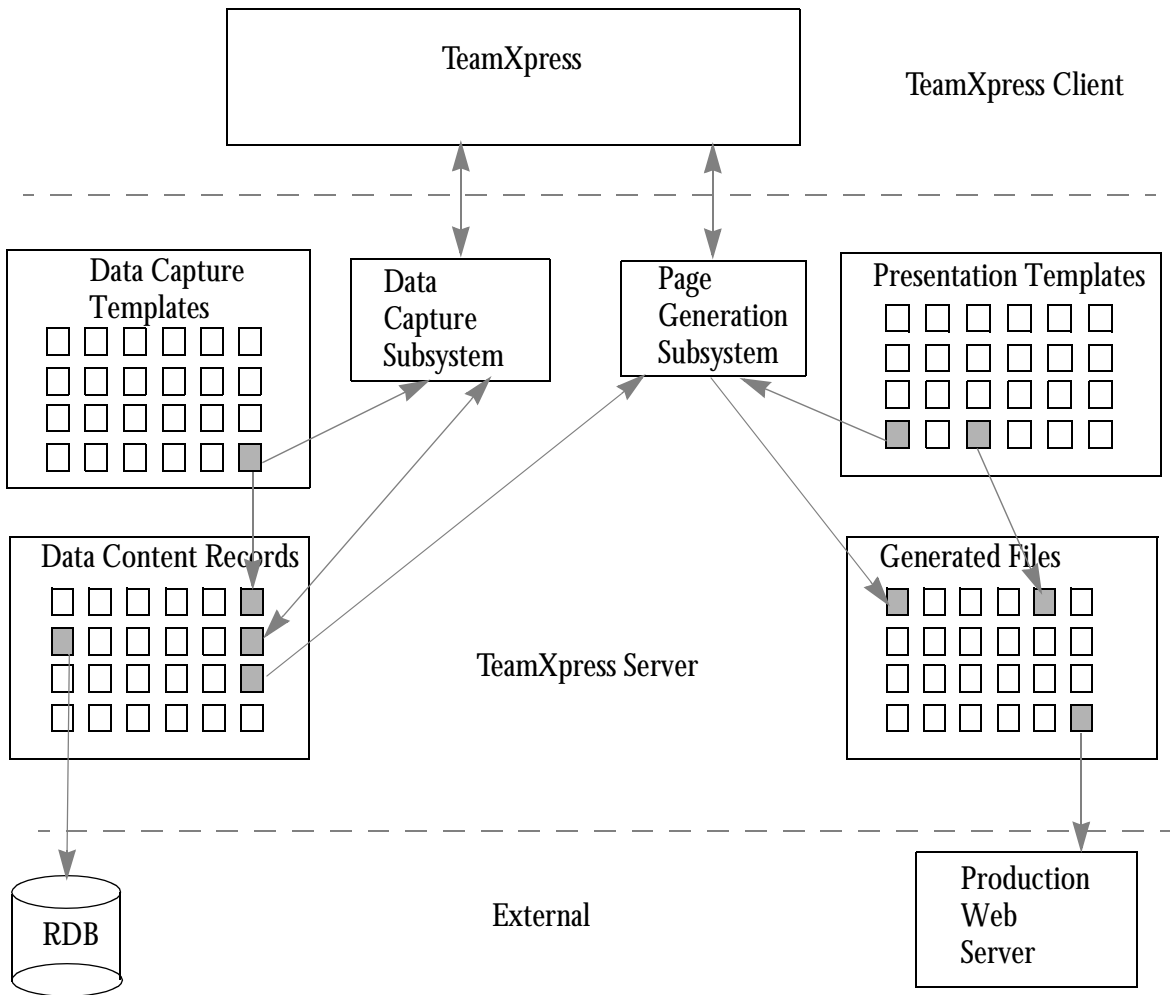
Configuring TeamXpress Templating consists of:

- Copying a set of example configuration files and directories supplied with TeamXpress Templating into specific locations in your system's directory structure. This sets up a fully functional example templating environment that lets you confirm that the TeamXpress Templating installation was successful and provides a default environment in which to familiarize yourself with TeamXpress Templating. See “Configuring the Example Templating Environment” on page 29 for more information.
- Customizing the templating environment for your specific site by renaming or creating new configuration files. See Chapter 3, “Setting Up Data Capture Templates.”

Concepts and Definitions

TeamXpress Templating Model

The TeamXpress Templating architecture allows data capture and data presentation to be configured, executed, and managed separately. The following diagram and sections provide a high-level overview of this architecture.



TeamXpress Templating Overview

Data Capture

Content contributors working through the TeamXpress GUI have access to the *data capture subsystem*. This subsystem lets content contributors select and work through forms defined by *data capture templates* to create or edit *data content records*, which by default are stored in the TeamXpress file system. Data is stored as XML and used later to fill in *presentation templates* to generate multiple renderings of the content, including for the web and wireless devices. After data content records are created, they can be displayed via presentation templates or optionally deployed to a database via DataDeploy.

Data Presentation

After data is captured and stored as data content records, users working through the TeamXpress Templating GUI, the TeamXpress GUI, or the command line can access the page generation subsystem to combine a data content record with a presentation template. The end result is a generated output file that displays the data content in a way defined by the presentation template. Additionally, users can generate an output file that obtains data from zero or one data content record and from queries to databases. The generated output file can optionally be deployed to a production web server via OpenDeploy.

Definitions

The following sections define key TeamXpress Templating terms.

Data Capture Template

A *data capture template* is an XML file named `datacapture.cfg` that defines the form used to capture data content from content contributors. A data capture template is associated with a category and type. The category and type define what type of data is required by the data capture template. The data that a content contributor enters in a data capture template is saved on the TeamXpress file system in the form of a data content record. See “Data Storage Hierarchy” on page 21 for information about where data capture templates reside.

Presentation Template

A *presentation template* is an XML file that defines how captured data will appear when displayed. A presentation template is populated with a data content record that was captured earlier (via a data capture template on the TeamXpress GUI) or from queries to databases. You can configure TeamXpress Templating to populate any presentation template with any data content record plus any additional information as required from an relational database. You can use presentation templates with component templates. A component template is a nested presentation template that is part of another presentation template. You can also use a single data content record to populate more than one presentation template, resulting in a different look and feel for the same data record. See “Data Storage Hierarchy” on page 21 for information about where presentation templates reside.

Data Content Record

A *data content record* is an XML file containing formatting information interspersed with data that was captured from a content contributor via the TeamXpress GUI. A data content record is named by the content contributor when it is saved.

Data Capture Subsystem

The *data capture subsystem* is a set of Java applications that perform the following functions:

- Read the `datacapture.cfg` and `templating.cfg` configuration files to determine what information should be presented via the TeamXpress GUI to a content contributor.
- Interpret content contributor input.
- Save content contributor input as formatted data content records.

Page Generation Subsystem

The *page generation subsystem* is a set of programs and libraries that perform the following functions:

- Read the presentation template and `templating.cfg` configuration files to determine what information should be presented to a content contributor via the TeamXpress GUI.
- Interpret content contributor input.
- Combine data content records and presentation templates to produce generated output files.

The presentation template compiler is the primary component of the page generation subsystem. It is a low-level command-line tool that invokes the template parser to create output files. The presentation template compiler is described in more detail in Appendix B, “Using Command-Line Tools.”



Configuration Files

TeamXpress Templating uses the following configuration files:

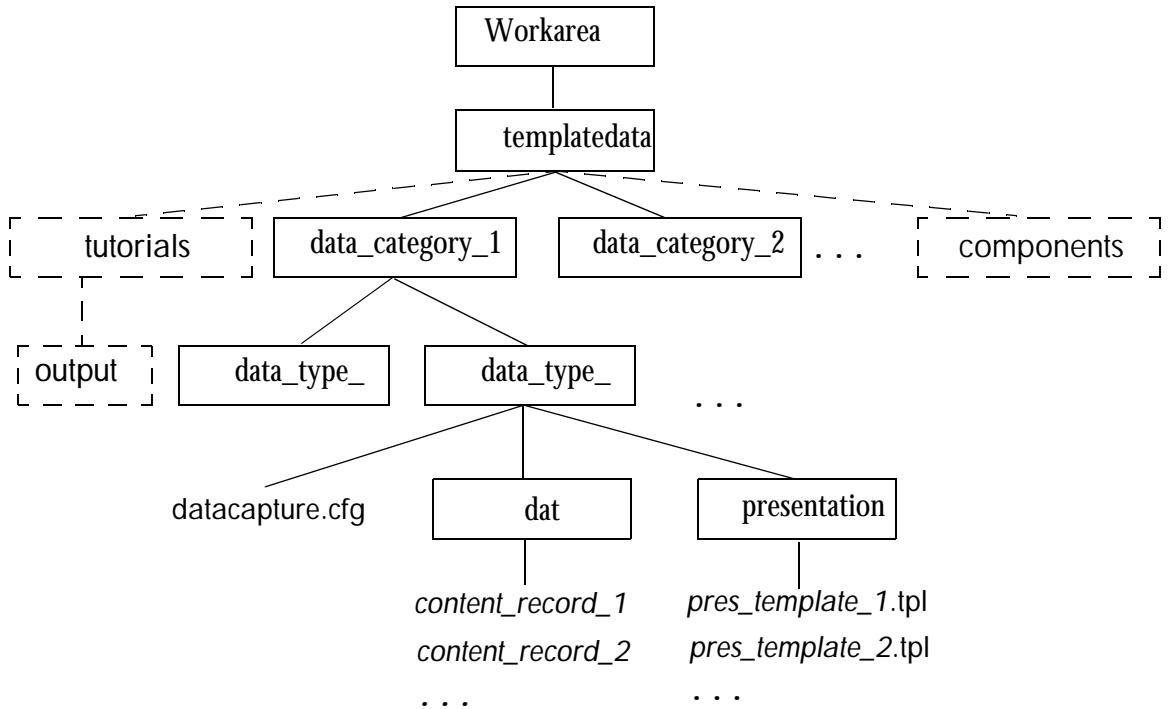
- `templating.cfg`: The main TeamXpress Templating configuration file. It is an XML file that resides outside of the TeamXpress file system in `iw-home/local/config` and specifies:
 - Which data categories and types are available for use with TeamXpress Templating.
 - Which presentation templates can generate HTML files on which TeamXpress branches and/or directories.
 - Which presentation templates can be used with a specific data type.
 - Which users or roles are allowed to create or edit data content records for a specific data type.
 - The location of the presentation template used for previewing generated HTML files.

See Chapter 5, “Mapping Users, Templates, and Content Records,” for details about customizing `templating.cfg`.

- `datacapture.cfg`: An XML file that defines a data capture template and drives data capture for a specific data type. As such, it defines the data type itself (i.e., what information the data type will contain, parameters that define what type of data is legal in any input field, etc.). A `datacapture.cfg` file also specifies the look and feel of the data capture form displayed in the TeamXpress GUI. A TeamXpress Templating environment can contain any number of `datacapture.cfg` files, differentiated from each other by where they reside in the directory structure. See “Data Storage Hierarchy” on page 21 for information about where `datacapture.cfg` files reside. See Chapter 3, “Setting Up Data Capture Templates,” for information about customizing `datacapture.cfg`.

Data Storage Hierarchy

TeamXpress Templating uses a data storage hierarchy based on data *categories* and *types*. The directory structure supporting this hierarchy resides in the workarea for each TeamXpress Templating user. The directory structure follows. Items in boxes are directories; items not in boxes are files.



TeamXpress Templating Directory Structure

The `templatedata` directory is at the highest level in the hierarchy.

Data categories are at the next level in the hierarchy and contain one or more data types. For example, the data category `beverages` could contain separate directories for the data types `tea`, `coffee`, `milk`, etc. In addition to residing in this directory structure, data categories and types must also be listed in the `templating.cfg` configuration file to be made available to TeamXpress Templating. See Chapter 5, “Mapping Users, Templates, and Content Records,” for more information. The `components` directory that stores component templates and the `tutorials` directory are optional subdirectories of `templatedata`.

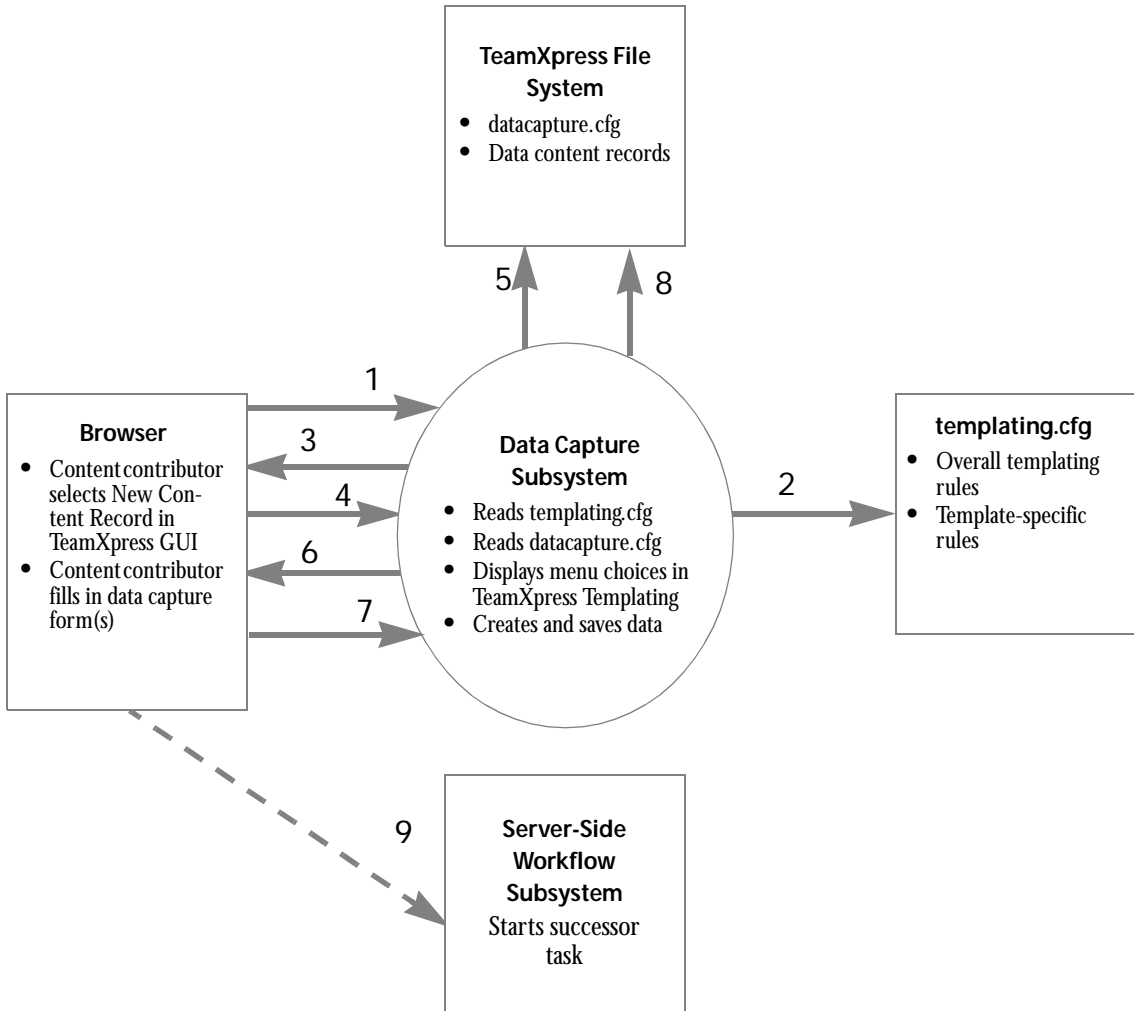
Data type directories each contain a `datacapture.cfg` file and the subdirectories `data` and `presentation`. Details for the entire hierarchy are as follows:

File or Directory	Description
<code>templatedata</code>	Top-level directory containing subdirectories for data categories, types, and all associated configuration files. Resides in the workarea for each user who uses TeamXpress Templating. Can be renamed and the <code>iw.cfg</code> file modified.
<code>data_category_1</code>	The first major categorization for data on a specific branch. Named and defined in <code>templating.cfg</code> . For example: <code>/templatedata/beverages</code>
<code>data_type_1</code>	The first subcategory of data in <code>data_category_1</code> . Named and defined in <code>templating.cfg</code> . For example: <code>/templatedata/beverages/tea</code> . Each data type in a given data category has its own subdirectory.
<code>datacapture.cfg</code>	The XML configuration file that defines a data capture template and drives data capture for a specific data type. As such, it defines the data type itself (i.e., what information the data type will contain, parameters for what type of data is legal in any input field, etc.). Specifies the look and feel of the data capture form displayed in the TeamXpress Templating GUI through which a content contributor enters data. Each data type must have exactly one <code>datacapture.cfg</code> file.
<code>data</code>	The directory containing all captured data content records for a given data type. If necessary, you can define and create a directory tree underneath the <code>data</code> directory. A <code>data</code> directory can contain zero or more data content records.

File or Directory	Description
content_record_1	The first data content record for a given data type. Each data content record is an XML file containing formatting information interspersed with data that was captured from a content contributor via the TeamXpress Templating GUI. A data content record is named by the content contributor during data entry. For example: /templatedata/beverages/tea/data/november_order
presentation	The directory containing all presentation templates for a given data type. The presentation directory must contain one or more presentation templates.
pres_template_1.tpl	The first presentation template for a given data type. A data type can have any number of presentation templates. A single presentation template is populated by data from zero or one data content record. A presentation template can have a name of your choice. For example: /templatedata/beverages/tea/presentation/monthly_order.tpl
components	The directory where all component templates are stored. This directory is not required or may be in another location.
tutorials	Examples showing the use of ix_xml tags. This directory is not required or may be in another location.
data_type_2	A second subcategory of data in data_category_1. For example: /templatedata/beverages/coffee
data_category_2	A second major categorization for data on a specific branch. For example: /templatedata/food

Process Flow: Creating a New Data Content Record

The following diagram shows the actions that take place when a content contributor creates a new data content record. Sections following the diagram explain each diagram step and component in detail.

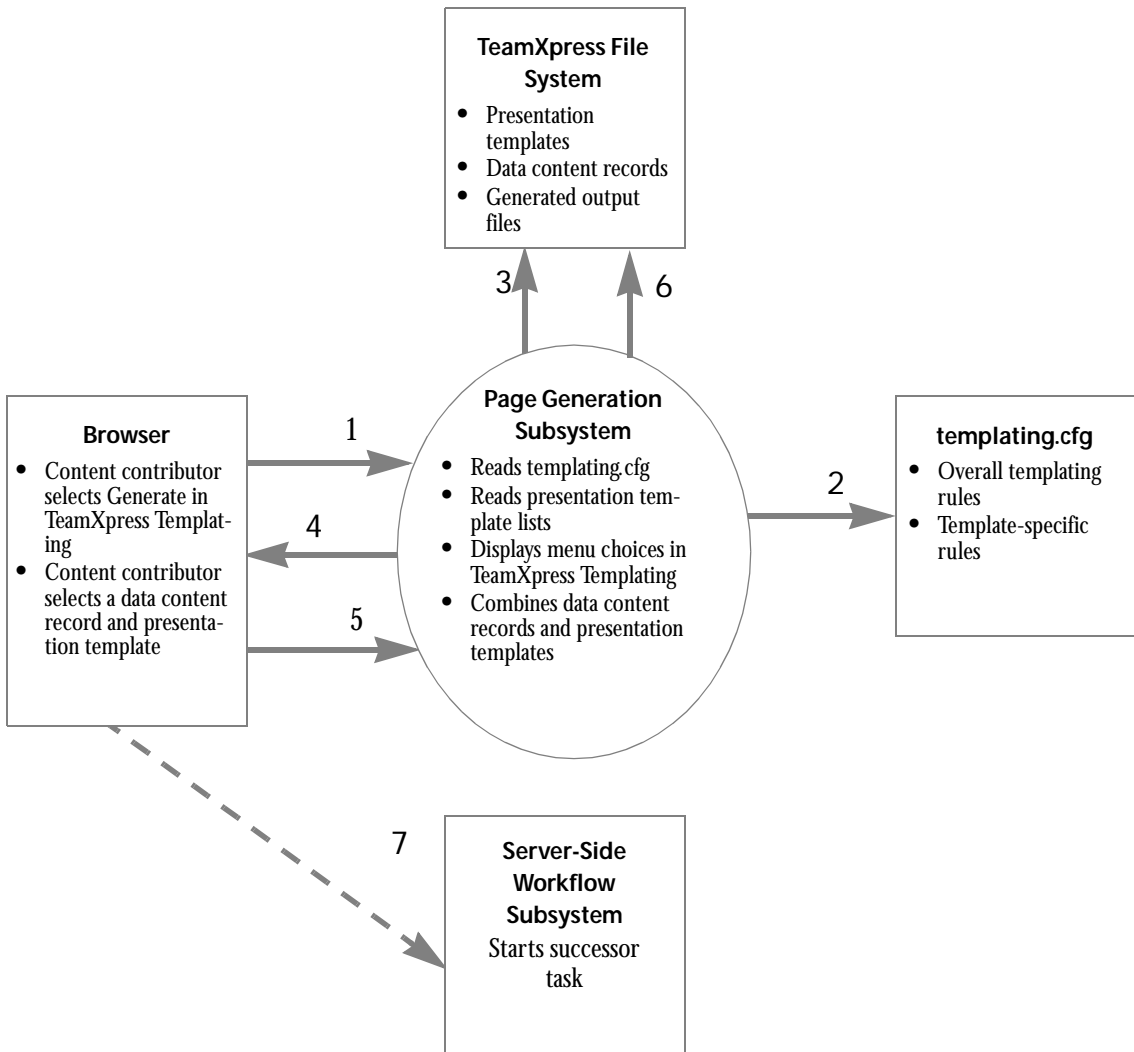


Process Flow Overview: Creating a New Data Content Record

1. A content contributor clicks **File > New Data Record** in the TeamXpress GUI.
2. TeamXpress Templating's data capture subsystem reads the `templating.cfg` file to determine which data types should be displayed in the TeamXpress Templating GUI as choices for the content contributor. The criteria used for this determination are specified in `templating.cfg` and can include the content contributor's login ID, role, or current TeamXpress area or branch. The data type must also exist as a directory in the content contributor's workarea.
3. The data capture subsystem displays the appropriate list of data categories and data types in a Create New Data Record dialog box in the TeamXpress Templating GUI.
4. The content contributor selects a data type. That information is sent back to the data capture subsystem.
5. The data capture subsystem reads the `datacapture.cfg` file for the data type chosen by the content contributor.
6. The data capture subsystem displays the data capture template (as defined by `datacapture.cfg`) in the data capture form window.
7. The content contributor enters data in the data capture template and selects **File > Save As** to name and save the data content record. The new data is sent to the data capture subsystem.
8. Using the data provided by the content contributor, the data capture subsystem writes a data content record to the TeamXpress file system. Note: The content contributor could also have chosen to preview the output file. In that situation, the data capture subsystem reads `templating.cfg` to determine which presentation templates are available for that data type. The content contributor selects a presentation template and the data capture subsystem displays a preview version of the data.
9. If creating the data content record is a task associated with a TeamXpress Workflow job, the user indicates the task has been completed and the TeamXpress workflow subsystem starts the successor task.

Process Flow: Generating an Output File

The following diagram shows the actions that take place when a content contributor generates a new output file by populating a presentation template with a previously captured data content record. Sections following the diagram explain each diagram step and component in detail.



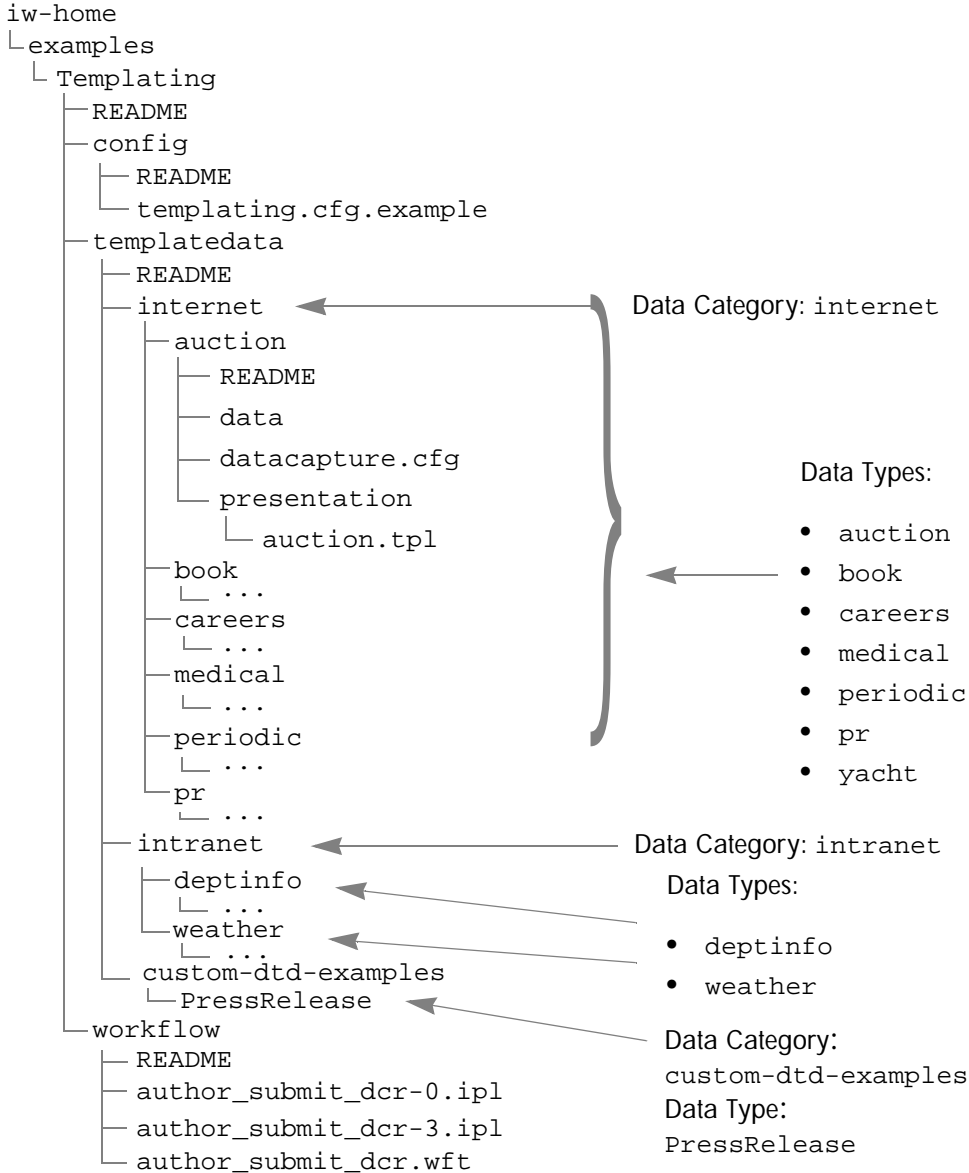
Process Flow Overview: Generating an Output File Using a Data Content Record and a Presentation Template

1. A content contributor clicks **File > Generate/Preview Page With** from the Templating menu.
2. TeamXpress Templating's page generation subsystem reads the `templating.cfg` file to determine which data content records for the selected data type should be displayed in the TeamXpress GUI as choices for the content contributor to choose from. The criteria used for this determination are specified in `templating.cfg` and can include the content contributor's login ID, role, or current TeamXpress area or branch. The user selects a data content record.
3. The page generation subsystem reads the `/templatedata/data_category/data_type/presentation` directory to determine which presentation templates are associated with the selected data type.
4. The page generation subsystem displays lists of the appropriate data content records and presentation templates in the Generate/Preview File window.
5. The content contributor selects a presentation template.
6. The page generation subsystem generates an output file by populating the chosen presentation template with data from the chosen data content record.
7. If creating the generated output file is a task associated with a TeamXpress Workflow job, the user indicates that task has been completed and the TeamXpress workflow subsystem starts the successor task.



The Example Directory Structure

The following directory structure is created when you install the TeamXpress Templating example:



Example TeamXpress Templating Directory Structure

The major components of the `iw-home/examples/Templating` directory structure are:

- A top-level `README` file.
- The `config` directory containing a `README` file and an example `templating.cfg` file.
- The `templatedata` directory containing a `README` file and three data category directories, `internet`, `intranet`, and `custom-dtd-examples`. The `internet` directory contains several data type directories (`auction`, `book`, etc.). Each data type directory contains a `datacapture.cfg` file, a `README` file, and the directories `data` and `presentation`. The `presentation` directory for each data type contains at least one presentation template file that generates an HTML file based on the data content records for that data type. Some data types have multiple presentation templates, in which case any of the presentation templates can be used for HTML file generation. The `custom-dtd-examples` directory contains an example using the DTD conversion procedures.
- The `workflow` directory containing a `README` file and all the files necessary to create the workflow job that deploys data content records via DataDeploy. The workflow template `author_submit_dcr.wft` defines the workflow job. The `.ipl` files define external tasks that are components of the job. The workflow job defined by these files executes automatically when an author creates and then submits a data content record to a staging area.

Configuring the Example Templating Environment

The following sections describe how to configure TeamXpress Templating to provide the example templating environment. After the initial setup is complete, you can:

- Use the example templating environment to become familiar with TeamXpress Templating's end-user features as described in the *TeamXpress User's Guide*.
- Customize the example templating environment as described in the remainder of this manual to create your site-specific configuration.

Perform the following steps to set up the example templating environment. You must copy most of these files and directories to locations that are specific to your site.

1. Decide which workarea you will use for the initial TeamXpress Templating setup. Ideally, this workarea should be on a temporary test branch where you can submit and publish without affecting the rest of your TeamXpress installation. After TeamXpress Templating is configured in the workarea on this test branch, you can copy the workarea to a permanent branch pertaining to your website. You can then submit the workarea to the staging area and then use **Get Latest** to propagate the setup to other workareas on the branch.
2. Read each directory's `README` file for details about directory contents and last-minute information that might not be documented elsewhere.
3. Copy the following files to the specified locations, ensuring that all users have read and write permission for each file except where noted otherwise:

Copy/rename this file:	To:
<code>iw-home/examples/Templating/config/templating.cfg.example</code>	<code>iw-home/local/config/templating.cfg</code> (writable only by system administrators)
<code>iw-home/examples/Templating/templatedata</code>	The workarea determined in Step 1. Copy the entire <code>templatedata</code> directory tree, including the <code>templatedata</code> directory itself. Do not change any directory or file names. The end result should be <code>workarea_name/templatedata/...</code>
The files <code>author_submit_dcr.wft</code> in the directory <code>iw-home/examples/Templating/workflow</code>	The <code>iw-home/local/config/wft/default</code> directory. The end result should be: <code>iw-home/local/config/wft/default/author_submit_dcr.wft</code>
The files matching the string <code>author_submit_dcr-*.ipl</code> in the directory <code>iw-home/examples/Templating/workflow</code>	The <code>iw-home/local/bin</code> directory. The end result should be: <code>iw-home/local/bin/author_submit_dcr-0.ipl</code> <code>iw-home/local/bin/author_submit_dcr-3.ipl</code>

4. Edit the `available_templates.ipl` file. See “Editing `available_templates.ipl` to Initiate Workflows” on page 31.

After you perform these tasks, the example templating environment is fully functional and integrated with TeamXpress workflow and DataDeploy. You can use the example templating environment to create or edit data content records, generate HTML files by combining a data content record with a presentation template, and deploy a data content record's extended attributes to a database via TeamXpress workflow and DataDeploy. See Chapter 6, "Integrating Templating, DataDeploy, and Workflow," for more information about integration with workflow and DataDeploy.

Editing `available_templates.ipl` to Initiate Workflows

The `available_templates.ipl` file contains a series of `elsif` statements that specify whether a particular event will be handled by a workflow. This file integrates workflow with templating. You only need to configure `available_templates.ipl` if you want the GUI to prompt the user to add data content records to a workflow on a create or delete event.

To configure `iw-home/local/config/wft/available_templates.ipl`, you must ensure that it contains at least one section for the `tt_data` command. The `tt_data` section should return a list of workflows relevant to creating a new data content record. The following is an example of a `tt_data` section to include in `available_templates.ipl`. This section is included in the TeamXpress Templating distribution in `iw-home/examples/Templating/workflow/README`.

```
elsif ($command eq "tt_data")
{
    if ($iw_role =~ m/author/i || $iw_role =~ m/editor/i) {
        return [
            [
                name => 'Author DCR Submit',
                file => 'default/author_submit_dcr.wft',
            ],
        ];
    }
}
```

This section says when authors or editors submit a data content record, they will be prompted to initiate a workflow job if:

- Creating the data content record was not already part of a workflow.
- Available workflow templates (WFTs) are defined in the `tt_data` section of the `available_templates.ipl` configuration file.

The workflow that will be used when a new data content record is submitted is `author_submit_dcr.wft`. If the `tt_data` section returns multiple workflows, the author or editor is prompted to select a workflow.

Use a `tt_deletedcr` section to specify the workflow that will be called when a data content record is deleted.

Modifying the TeamXpress `iw.cfg` File

This section describes some options that may need to be set in the `[teamsite_templating]` section of the TeamXpress `/etc/iw.cfg` file.

Saving Preview Files

Previewed templating files are stored in the `preview-dir` directory. By default, previewed files that have not been modified in the last 60 minutes are deleted when a preview is performed. You can change the length of time for deleting files in the `/etc/iw.cfg` file as follows (where `value` is a number representing the number of minutes files are to remain in the preview directory):

```
[teamsite_templating]
preview_file_max_age=value
```

Identifying the Templating Directory

If you need to change the directory in your workareas where templating content will reside, you can modify the `/etc/iw.cfg` file. The default directory is `/"templatedata"`.

```
[teamsite_templating]
data_root="/"directory"
```

Identifying the Templating Interface

By default, TeamXpress Templating 1.0 uses the browser-based interface for displaying the data capture form. If you want to use the new Java-based interface, include the following line in the `/etc/iw.cfg` file. This manual describes the features of the Java-based interface.

```
[teamsite_templating]
use_java_ui=true
```

Identifying the Validation Regex

By default, TeamXpress Templating uses basic regex(5) for validation. When using the Java-based user interface, you should use extended validation regex. Including the following information in the `/etc/iw.cfg` file.

```
[teamsite_templating]
use_extended_regex5=true
```

Identifying the Preview Directory

You can control the location of preview files by including the following lines in your `/etc/iw.cfg` file.

```
[teamsite_templating]
use_preconnect_remap=true
```

When `use_preconnect_remap=true`, the preview file is placed in the `templatedata/iw_preview` directory. The proxy server configuration (page 34) is relevant. The actual preview file is placed in the preview directory specified in `templating.cfg`. Each presentation template has a preview attribute. The default file is called `zz_tst_temp_preview.*`, where the extension is determined by the extension attribute of the presentation template. The templating preview file name can be changed by the flag `preview_file_name`:

```
[teamsite_templating]
preview_file_name=filename
```

When the preview file name is changed, users should be notified of the new file name.

When `use_preconnect_remap=false`, no file cleanup occurs because the files exist outside of safe directory boundaries. Users should manually remove `zz_tst_temp_preview.*` files (or files specified by the `preview_file_name` flag) from their workarea. The `iwproxy` server configuration (page 34) is not relevant.

Adding DCR Search to the View Menu

If DataDeploy's Database Auto Synchronization has been set up, the data content record search feature is available. You need to uncomment the following line in `/etc/iw.cfg` to add the **Search Data Records** menu item to the **TeamXpress View** menu.

```
#custom_menu_item_searchdcr="View", "Search Data Records",  
"iwsearchdcr.cgi", "all", "scrollbars=yes,resizable=yes",  
width=640,height=545"
```

Refer to Chapter 6 of the *TeamXpress Administration Guide* for information on metadata capture and search.

Proxy Server Configuration

The TeamXpress Templating installation procedure automatically enables template previewing by adding the following line to the `[iwproxy_preconnect_remap]` section of `/etc/iw.cfg`:

```
_regex=(.*/WORKAREA/[^/]+)/.*\?iw_dataroot\=(.*)&iw_key\=(.*)=$1/$2/  
iw_preview/$3
```

Under normal circumstances, you do not need to add this line manually. It is shown here in case you need to verify its existence or accuracy in `iw.cfg`.

Starting TeamXpress Templating

Perform the following steps to start TeamXpress Templating after you have configured the example templating environment:

1. Log out of TeamXpress.
2. Log back into TeamXpress.
3. Select **File > New Data Record**.
4. As prompted, install the client module for TeamXpress Templating.

The example templating environment should now be accessible via the Java-based TeamXpress Templating GUI as described the *TeamXpress User's Guide*.



Setting Up Data Capture Templates

This chapter describes how to edit and create data capture templates. It is assumed that the example templating environment's directory structure already exists on your system and that you now intend to customize this environment by creating new data capture templates. See Chapter 2, "Initial Configuration," for more information about the example templating environment's directory structure.

This chapter contains:

- An overview of data capture templates.
- Pointers to sample data capture template files that are included with this release of TeamXpress Templating
- Examples of data capture forms and the data capture template files that generate them.
- A sample data content record.
- The data capture template document type definition (DTD).

You may also create data capture template files from industry-standard XML DTDs. Refer to Appendix A, "Creating Data Capture Templates from DTDs."

Data Capture Template Overview

Data capture templates are XML files named `datacapture.cfg` that reside in the locations described in “Data Storage Hierarchy” on page 21. Each `datacapture.cfg` file contains the following components:

- *Rule set*: A set of configuration instructions that controls the appearance and behavior of the data capture forms displayed in the TeamXpress GUI. A TeamXpress Templating `datacapture.cfg` file must contain exactly one rule set. Each rule set contains one or more *of the elements identified by the `%items` parameter entity reference (currently `item` or `container`)*.
- *Item*: Each item is a single set of data that is to be captured from a content contributor. A rule set must contain at least one item. Items can be nested within other items. If a rule set contains more than one item, item names must be unique within any given nesting level. See page 44 for more information. Each item contains one or more *instances*.
- *Instance*: Each instance defines how to capture data for an item. An instance also defines an ACL that determines which if any instance a specific user is allowed to use to enter the data. See page 46 for more information.

The following list describes the characteristics of data capture forms that you can configure in a `datacapture.cfg` file. Additional customization is not available.

- The number and appearance of data capture fields in a data capture form.
- The content and appearance of labels for each data capture field in a data capture form.
- How data will be captured, such as through a check box, radio button, text field, etc.
- Characteristics of the data entered in each section’s fields, such as text style (such as bold or underline), hypertext link, maximum length, whether the data is text or image.
- Which fields, if any, must be filled in before the data entry form can be saved.
- Which fields can be filled in by a specific content contributor.
- Which data entry fields can be displayed multiple times in the same data capture form, and how many times the field can be displayed.

When a content contributor finishes filling in a data capture form and selects **File > Save**, the data capture subsystem combines the newly entered data with the XML rules defined in the `datacapture.cfg` rule set(s). The end result is a data content record that is an XML file that associates field names from the data capture form with the values that were entered in those fields by the content contributor.

Data capture templates should validate against the `datacapture4.5.dtd` file, which can be found at `iw-home/local/config/datacapture4.5.dtd`.

Example Data Capture Templates

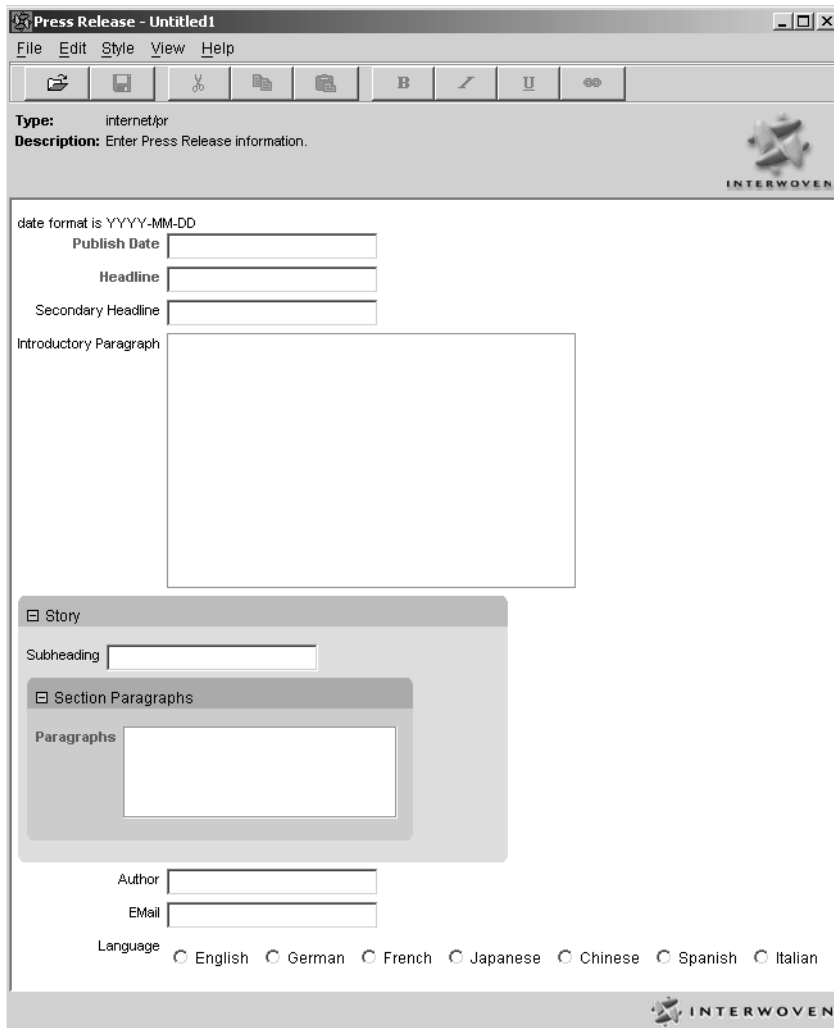
TeamXpress Templating ships with an extensive set of example data capture templates that are available for use in the example templating environment. See “Configuring the Example Templating Environment” on page 29 for descriptions and locations. Some of these templates are described in this section.

Data Capture Example 1

The following sections show a hypothetical Press Release data capture form, the `datacapture.cfg` file that generates it, and the data content record that is created when the form is saved.

Example 1 Data Capture Form

The following is a hypothetical Press Release data capture form. This form is included in the TeamXpress Templating distribution and is available for use after you configure the example templating environment.



Press Release - Untitled1

File Edit Style View Help

Type: internet/pr
Description: Enter Press Release information.

date format is YYYY-MM-DD

Publish Date

Headline

Secondary Headline

Introductory Paragraph

Story

Subheading

Section Paragraphs

Paragraphs

Author

Email

Language English German French Japanese Chinese Spanish Italian

Press Release Data Capture Form (without data)

Example 1 datacapture.cfg File

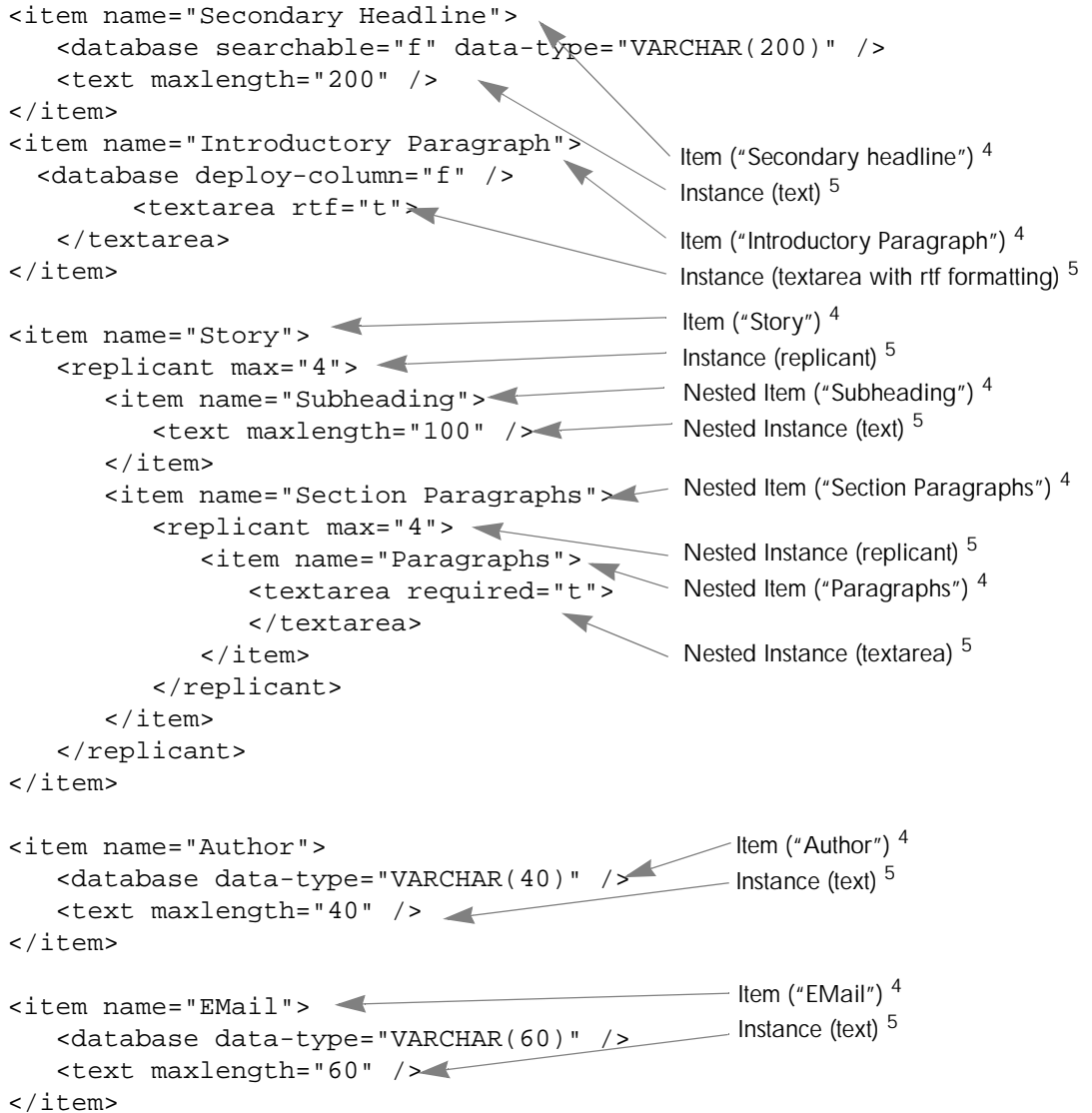
The datacapture.cfg file that generates this Press Release data capture form is shown below. Like all datacapture.cfg files, it consists of a rule set, items, and instances. See “Diagram Key” on page 43 for an explanation of each referenced item. For details about additional datacapture.cfg features not illustrated by this example, see the DTD starting on page 63. An additional sample file specific to TeamXpress metadata capture is located in the *TeamXpress Administration Guide*. While the syntax for the metadata capture version of datacapture.cfg differs slightly from the TeamXpress Templating version, it is similar enough to provide a useful detailed example. It is recommended that you refer to that example in addition to the following one.

```
<?xml version="1.0" encoding = "UTF-8"? standalone="no"?> DCT Identifier 1
<!DOCTYPE datacapture SYSTEM "datacapture4.5.dtd">

<data-capture-requirements type="content" name="pr">
  <!-- data-capture-requirements elements contain area elements -->
  <ruleset name="Press Release"> 2 ← Rule Set ("Press Release")
    <description> 3 ← Description
      Enter Press Release information.
    </description>

    <item name="Publish Date"> 4 ← Item ("Publish Date") with
      <description>date format is YYYY-MM-DD 4 ← description and
      </description> 4 ← database elements
      <database data-type="DATE" data-format="YYYY-MM-dd"> 5 ← Instance (text)
      </database> 5 ← with validation
      <text required="t" maxlength="10" 5 ← regex
        validation-regex="^[0-9][0-9][0-9][0-9]-[0-1]
          [0-9]-[0-3][0-9]$" />
    </item>

    <item name="Headline"> 4 ← Item ("Headline")
      <database data-type="VARCHAR(100)" /> 5 ← Instance (text)
      <text required="t" maxlength="100" />
    </item>
```



```

<item name="Languages">
  <database data-type="VARCHAR(10)" />
  <radio>
    <option label="English" value="English" />
    <option label="German" value="German" />
    <option label="French" value="French" />
    <option label="Japanese" value="Japanese" />
    <option label="Chinese" value="Chinese" />
    <option label="Spanish" value="Spanish" />
    <option label="Italian" value="Italian" />
  </radio>
</item>

</ruleset>
</data-capture-requirements>

```

Diagram Key

- DCT Identifier:** The `<data-capture-requirements>` element lets you assign a unique identifier for each data capture template. Exactly one `<data-capture-requirements>` element is required in all `datacapture.cfg` files. The `name` attribute within a `<data-capture-requirements>` element is optional. The `type` attribute values—`content`, `metadata`, and `workflow`—let you further describe the type of data that will be captured by the template. For data capture templates, `content` should be specified. The information in a `<data-capture-requirements>` element is for reference only. None of the information in this element is stored in the data content record that is created when `datacapture.cfg` is processed by the data capture subsystem.
- Rule Set:** The `<ruleset>` element contains all of the items that make up the rule set that defines the appearance and behavior of the data capture form. A `datacapture.cfg` file must contain exactly one `<ruleset>` element. The `name` attribute within a `<ruleset>` element is also required. The value of the `name` attribute appears in the TeamXpress GUI as the name of the data capture form (**Press Release** in this example). Optional subelements are `<label>`, `<description>` (see number 3), and `(%items;)` (see number 4). The `<label>` subelement is used to provide a label on the data capture form. The parameter entity reference `(%items;)` is currently either `<item>` or `<container>`. A container is a non-repeating, named set of data capture items.

A `<container>` may appear anywhere in a data capture template that an `<item>` element may appear. A container is conceptually similar to an item with a replicant of `min = 1` and `max = 1`, but it is more efficient.

- 3. Description:** The optional `<description>` subelement inserts a description in the data capture form. A `<description>` subelement can reside anywhere inside the `<ruleset>` element as a child element of `<ruleset>`.
- 4. Item:** The `<item>` element assigns a name of your choice to an item and contains the instances and/or other nested items that specify how to capture data for the item. A `<ruleset>` element can contain any number of `<item>` elements. Each `<item>` element must contain at least one instance. The optional subelements for `<item>` are `<label>`, `<description>`, and `<database>`. The `<label>` and `<description>` subelements consist of character data. The information provided by `<label>` is used as the field name in the DCT. If `<label>` is not included, the `name` attribute of the `<item>` element is used as the field name. A `<description>` provides more details about what the data capture item represents or the format that may be required for data entry.

The `<database>` subelement facilitates the use of the appropriate data type in DataDeploy and does not impact templating. The `<database>` subelement has four attributes: `deploy_column` specifies whether a column in the DataDeploy table should be built for that item; `searchable` can be either "t" (default) or "f"; `data-type` is required and is any valid JDBC database type; `data-format` describes the format if `date` or `time` is specified for the `data-type` attribute. If a value for `data-format` is specified, the instance should contain a validation regex to force the correct entry in the field.

Item names must be unique within a nested section. For example, the following syntax is *illegal* because it uses the item name `Section` twice in the same nested section in the `<ruleset>` element:

```
<ruleset name="Press Release"
  <item name="Section">
    <text size="40" maxlength="100">
    </text>
  </item>
  <item name="Section">
    <text size="80" maxlength="200">
    </text>
  </item>
</ruleset>
```

However, the following syntax is legal because it uses the item name `Section` in different nested sections:

```
<ruleset name="Press Release">
  <item name="Morning Edition">
    <replicant required="t" max="4">
      <item name="Section">
        <text size="80" maxlength="200">
        </text>
      </item>
    </replicant>
  </item>
  <item name="Evening Edition">
    <replicant required="t" max="4">
      <item name="Section">
        <text size="100" maxlength="400">
        </text>
      </item>
    </replicant>
  </item>
</ruleset>
```



5. **Instance:** An instance defines how to capture data for an item. An instance can also define an ACL that determines which (if any) instance a specific user is allowed to use to enter data. Instances can be any of:

Instance	Description
<browser>	<p>Lets a content contributor navigate through the workarea to select a file.</p> <p>Attributes:</p> <ul style="list-style-type: none">• <code>ceiling-dir</code>: Sets the upper boundary for navigation. The content contributor can never go above the current workarea in the directory structure. The <code>ceiling-dir</code> attribute lets you set the ceiling below the current workarea.• <code>extns</code>: Comma delimited list of file extensions. Files having these extensions are displayed during navigation.• <code>initial-dir</code>: The initial directory displayed at the start of navigation.• <code>size</code>: The number of characters that can display in the browse field.• <code>maxlength</code>: The maximum number of characters the user can enter.• <code>required</code>: Specifies whether data must be captured by this instance. The default setting is <code>f</code> (not required). Setting it to <code>t</code> specifies that a user must specify a value for this item.

Instance	Description
<p><browser> (continued)</p>	<p>Subelements:</p> <ul style="list-style-type: none"> • <allowed>: Lets you set an ACL to specify which users can or cannot use a specific instance to enter data. If <allowed> is not set, any user can enter data for the instance. The <allowed> element can have any of the following subelements: <ul style="list-style-type: none"> - <cred>: Lets you name a user or role in the ACL (e.g., user="joe" or role="master"). - <and>: Logical and statement for grouping ACL credentials. - <or>: Logical or statement for grouping ACL credentials. - <not>: Logical not statement for negating ACL credentials. Users who are not allowed do not see the instance on their data capture form. <p>See the examples on page 53.</p> • <callout>: Creates a button that calls a Java external program (see page 54). <ul style="list-style-type: none"> - type: Must be "java-class". - location: Specifies the URL of a jar file or class file. The file does not necessarily have to be on the same server as TeamXpress Templating. - class: Specifies the actual name of the class in the jar file. - label: Label of the button that launches the callout code.



Instance	Description
<checkbox>	<p>Specifies that data will be captured via one or more check boxes.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>delimiter</code>: Specifies the delimiting character used when data from all check boxes is concatenated by the data capture subsystem. The default delimiter is a comma (,). • <code>required</code>: See <browser> above. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><callout></code>: See <browser> above. • <code>(%chooser-options;)</code>: Parameter entity reference that has the following values: <ul style="list-style-type: none"> - <code><inline></code>: Provides a method for making server side inline callout programs that return multiple XML elements to the data capture form (see page 53 for additional details). - <code><option></code>: Lets you assign a label or value to a check box so that a user can enter only the predetermined label or value data by checking the check box. Also lets you specify whether the check box is initially displayed as being checked by default. A <checkbox> element must have at least one <option> subelement. See the DTD on page 63 for syntax details.
<hidden>	<p>Specifies that the data will not be shown in the data capture form. A <hidden> field may receive data from a callout program.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>required</code>: See <browser> above. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><callout></code>: See <browser> above.

Instance	Description
<radio>	<p>Specifies that data will be captured via one or more radio buttons.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>required</code>: See <browser> above. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><callout></code>: See <browser> above. • <code><inline></code>: See <checkbox> above. • <code><option></code>: See <checkbox> above. A <radio> element must have at least one <option> subelement.
<readonly>	<p>Specifies that the data will be shown on the data capture form but will not be editable.</p> <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><callout></code>: See <browser> above.



Instance	Description
<replicant>	<p>Specifies a repeatable instance that can contain multiple nested items and instances. When there are multiple instances, the first instance whose ACL allows the current user to enter data will be the instance used for that user. <replicant> is the only instance that can contain nested items and instances. Whenever additional iterations of the instance can be displayed (i.e., if the <code>max</code> threshold has not yet been reached), the Edit >Insert Above and Edit >Insert Below menu items are active. Whenever iterations of the instance can be removed (i.e., if the <code>min</code> threshold has not yet been reached), the Edit >Delete menu item is active. If a <replicant> has four items, the Insert menu item displays another set of four items in the data capture form.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>default</code>: The number of instance iterations displayed initially in the data capture form. • <code>max</code>: The maximum number of items that can reside within the replicant instance. • <code>min</code>: The minimum number of items that can reside within the replicant instance. • <code>combination</code>: Specifies whether the entire set of items will be replicated when the user requests a replicant or whether the user will be prompted to select one of the replicant items. • <code>hide-name</code>: Determines whether the label displays for each replicant. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><item></code>: See Item on page 44. • <code><container></code>: A <code><container></code> is a non-repeating, named set of data capture items. In addition to the <code>combination</code> and <code>hide-name</code> attributes, you can also include the <code>name</code> attribute to specify the field name.

Instance	Description
<select>	<p>Specifies that data will be captured via a drop-down list.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>delimiter</code>: See <checkbox> above. • <code>required</code>: See <browser> above. • <code>multiple</code>: Specifies whether more than one item can be selected. The default value is <code>f</code> (only one item can be selected). Setting <code>multiple="t"</code> specifies that a user can select more than one item. • <code>size</code>: The number of selections that display in the selection box at one time. • <code>width</code>: The width of the drop-down or select list. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><callout></code>: See <browser> above. • <code><inline></code>: See <checkbox> above. • <code><option></code>: See <checkbox> above.
<text>	<p>Specifies that data will be entered and captured via an unformatted text field.</p> <p>Attributes:</p> <ul style="list-style-type: none"> • <code>maxlength</code>: The maximum number of characters the user can enter. • <code>required</code>: See <browser> above. • <code>size</code>: The number of characters that display in the text box. • <code>validation-regex</code>: Uses Perl regex syntax to set validation criteria for text entered by a user. A retry message is displayed in the data capture form if the entered text does not meet the specified criteria. <p>Subelements:</p> <ul style="list-style-type: none"> • <code><allowed></code>: See <browser> above. • <code><default></code>: The default text that displays in the field when the data capture form opens. • <code><callout></code>: See <browser> above.



Instance	Description
<code><textarea></code>	<p>Specifies that data will be entered and captured via a text field of a specified size.</p> <p>Attributes:</p> <ul style="list-style-type: none">• <code>cols</code>: The width (in characters) of the text area. If <code>rtf="t"</code> is set, width in pixels.• <code>required</code>: See <code><browser></code> above.• <code>rows</code>: The height (in rows) of the text area. If <code>rtf="t"</code> is set, height in pixels.• <code>wrap</code>: Handles word wrapping in text input areas in forms. When <code>off</code> is set, lines are sent exactly as typed; when <code>virtual</code> is specified, the text word wraps in the form, but long lines are sent as one line; when <code>physical</code> is set, the word wraps and text are transmitted at all wrap points.• <code>validation-regex</code>: See <code><text></code> above.• <code>rtf</code>: Allows user to provide text styles such as bold, italics, and underscoring. <p>Subelements:</p> <ul style="list-style-type: none">• <code><allowed></code>: See <code><browser></code> above.• <code><default></code>: See <code><text></code> above.• <code><callout></code>: See <code><browser></code> above.

Details on Attributes and Subelements of Instances

This section provides additional details or examples on the attributes and subelements described in the table of instances.

The `<Allowed>` Attribute

The following code would allow all users except `joe` to use the current instance:

```
<allowed>
  <not>
    <cred user="joe">
    </cred>
  </not>
</allowed>
```

In the following example, `<allowed>` would set an instance that only editors can use while another instance is available for all other roles. The first instance a user satisfies is the one that is used.

```
<item name= "abc">
  <instance>
    <!--only for editors-->
    <allowed> <cred role="editor"/> </allowed>
  </instance>
  <instance>
    <!--for everyone else-->
  </instance>
</item>
```

The `<inline>` Subelement

An `<inline>` element should have a `command` attribute such as:

```
<inline command="/bin/cat /tmp/a /tmp/b"/>
```

The inline callout program should return a well-formed XML document. The document's outermost element should be a `<substitution>` element. It should contain any XML that is valid according to `datacapture4.5.dtd`. That `<substitution>` element will contain six `<option>` elements, enumerating a variety of types of yacht hull materials (see page 59).



```
<?xml version="1.0" encoding="UTF-8"?>
<substitution>
  <option value="Lead" label="Lead"/>
  <option value="Tin" label="Tin"/>
  <option value="Silicon" label="Silicon"/>
  <option value="Plastic" label="Plastic"/>
  <option value="Paper" label="Paper"/>
  <option value="Glass" label="Glass"/>
</substitution>
```

This simple callout output a static result. A more sophisticated callout program could query a database and return the query results as `<option>` elements.

The `<Callout>` Subelement

The `<callout>` subelement creates a button on the data capture form that can be programmed to call a Java program. An interface is provided that declares the `IWDataCaptureCallout` interface. You need to write a Java class that implements the interface. Java documentation (javadoc) that describes the API is available once you install TeamXpress Templating. You can access this Javadoc through a browser at <http://TeamXpress-server/iw/java-callout-api/tree.html>. Source code and example classes can be accessed at `iw-home/local/config/java-callout-api`.

Example 1 Data Content Record

This section shows the data content record that is created if a content contributor enters the following data in the Press Release data capture form:

Press Release - Untitled1 [Modified]

File Edit Style View Help

Type: internet/pr
Description: Enter Press Release information.

date format is YYYY-MM-DD

Publish Date: 2000-08-24

Headline: Candidate Joins Race

Secondary Headline:

Introductory Paragraph:

Story

Subheading:

Section Paragraphs

Paragraphs: A new candidate enters the r...

Author: eal

EMail: eal@example.com

Language: English German French Japanese Chinese Spanish Italian

Press Release Data Capture Form (with data)

The resulting data content record is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE record SYSTEM "dcr4.5.dtd">
<record name="eal.pr.1" type="content">
  <item name="Publish Date">
```



```
<value>2000-08-28</value>
</item>
<item name="Headline">
  <value>Candidate Joins Race</value>
</item>
<item name="Secondary Headline">
  <value></value>
</item>
<item name="Introductory Paragraph">
  <value></value>
</item>
<item delimiter=", " name="Story">
  <value>
    <item name="Subheading">
      <value></value>
    </item>
    <item name="Section Paragraphs">
      <value>
        <item name="Paragraphs">
          <value>A new candidate entered the race as of 8/28/00.
          </value>
        </item>
      </value>
    </item>
  </value>
</item>
<item name="Author">
  <value>eal</value>
</item>
<item name="EMail">
  <value>eal@example.com</value>
</item>
<item name="Languages">
  <value>English</value>
</item>
</record>
```

Data Capture Example 2

The following sections show a hypothetical Yacht Information data capture form and the `datacapture.cfg` file that generates it.

Example 2 Data Capture Form

The following is a hypothetical Yacht Information data capture form. This form is included in the TeamXpress Templating distribution and is available for use after you configure the example templating environment.



Vessel Information - Untitled2

File Edit Style View Help

internet/yacht

Description: Allows the entry of data relating to a sailing vessel and its seasonal charter prices.

Boat Manufacturer

Boat Model

Picture Browse ...

length in feet, range 0-999

Length

Rig

Hull Type Monohull Catamaran Trimaran

Hull Material

Pricing

Season
Winter
Spring
Summer
Fall

Time Periods

Time Period

Price

Number of Cabins

Number of Staterooms

Spinnaker Included

Tri-sail Included

Genoa Included

Jib Included

Storm Jib Included

Dinghy Included

Liferaft Included

EPIRB Included

Details

Example 2 datacapture.cfg File

The datacapture.cfg file that generates this Yacht Information data capture form is as follows:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE datacapture SYSTEM "datacapture4.5.dtd">

<data-capture-requirements type="content" name="yacht">
  <!-- data-capture-requirements elements contain area elements -->
  <ruleset name="Vessel Information">

    <description>
      Allows the entry of data relating to a sailing vessel and its
      seasonal charter prices.
    </description>

    <item name="Boat Manufacturer">
      <database data-type="VARCHAR(40)" />
      <text required="t" maxlength="40" />
    </item>

    <item name="Boat Model">
      <database data-type="VARCHAR(40)" />
      <text required="t" maxlength="40" />
    </item>

    <item name="Picture">
      <database deploy-column="f" />
      <browser extns=".gif,.jpg"
        initial-dir="/templatedata/internet/yacht/images"/>
    </item>

    <item name="Length">
      <database data-type="SMALLINT" />
      <text required="t" maxlength="3" validation-regex="^[0-9]\\{0,\\}$" />
    </item>

    <item name="Rig">
      <database data-type="CHAR(6)" />
      <select required="t">
        <option selected="t" value="Sloop" label="Sloop"/>
        <option value="Ketch" label="Ketch"/>
        <option value="Cutter" label="Cutter"/>
      </select>
    </item>

    <item name="Hull Type">
```



```
<database data-type="CHAR(9)" />
<radio required="t">
  <option selected="t" value="Monohull" label="Monohull"/>
  <option value="Catamaran" label="Catamaran"/>
  <option value="Trimaran" label="Trimaran"/>
</radio>
</item>

<item name="Hull Material">
  <database data-type="VARCHAR(15)" />
  <select required="t">

    <!-- To use the example server-side
         inline callout, uncomment the
         next line for Solaris:

    <inline command="__IW_HOME__/iw-perl/bin/iwperl __IW_HOME__/examples/
Templating/config/example_server_side_inline_callout.ipl" />

        or this line for Windows NT/2000:

    <inline command="__IW_HOME__/iw-perl/bin/iwperl.exe __IW_HOME__/examples/
Templating/config/example_server_side_inline_callout.ipl" />

    replacing "__IW_HOME__" with
    the location of your TeamXpress
    installation. -->

    <option value="Fiberglass" label="Fiberglass"/>
    <option value="Wood" label="Wood"/>
    <option value="Steel" label="Steel"/>
    <option value="Aluminium" label="Aluminium"/>
    <option value="Ferrocement" label="Ferrocement"/>
    <option value="Other" label="Other"/>
  </select>
</item>

<item name="Pricing">
  <database deploy-column="f" />
  <replicant min="1" max="5">
    <item name="Season">
      <select required="t" multiple="t" size="5">
        <option selected="t" value="All Year" label="All Year"/>
        <option value="Winter" label="Winter"/>
        <option value="Spring" label="Spring"/>
      </select>
    </item>
  </replicant>
</item>
```

```

        <option value="Summer" label="Summer" />
        <option value="Fall" label="Fall" />
    </select>
</item>

<item name="Time Periods">
    <replicant min="1" max="3">
        <item name="Time Period">
            <select required="t">
                <option value="Day" label="Day" />
                <option value="Week" label="Week" />
                <option value="Month" label="Month" />
            </select>
        </item>

        <item name="Price">
            <text required="t" />
        </item>
    </replicant>
</item>
</replicant>
</item>

<item name="Number of Cabins">
    <database data-type="SMALLINT" />
    <select required="t">
        <option value="1" label="One" />
        <option value="2" label="Two" />
        <option value="3" label="Three" />
        <option value="4" label="Four" />
        <option value="5" label="Five" />
        <option value="6" label="Six" />
    </select>
</item>

<item name="Number of Staterooms">
    <database data-type="SMALLINT" />
    <select required="t">
        <option value="1" label="One" />
        <option value="2" label="Two" />
        <option value="3" label="Three" />
        <option value="4" label="Four" />
        <option value="5" label="Five" />
        <option value="6" label="Six" />
    </select>

```



```
</item>

<item name="Spinnaker">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Tri-sail">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Genoa">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Jib">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Storm Jib">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Dinghy">
  <database data-type="CHAR(3)" />
  <checkbox>
    <option value="Yes" label="Included"/>
  </checkbox>
</item>

<item name="Liferaft">
```



```

        <database data-type="CHAR(3)" />
        <checkbox>
            <option value="Yes" label="Included"/>
        </checkbox>
    </item>

    <item name="EPIRB">
        <database data-type="CHAR(3)" />
        <checkbox>
            <option value="Yes" label="Included"/>
        </checkbox>
    </item>

    <item name="Details">
        <database deploy-column="f" />
        <textarea/>
    </item>

</ruleset>
</data-capture-requirements>

```

Data Capture Template DTD

The following code shows the `datacapture4.5.dtd` file that contains the syntax of the elements needed to create a `datacapture.cfg` file.

```

<!-- Start with some basic parameter entities. -->
<!ENTITY % data-capture-requirements-contentspec "ruleset*">
<!ENTITY % items "container|item">
<!ENTITY % chooser-options "option|inline">

<!-- The next element type is specific to datacapture4.5.dtd. -->

<!-- An 'inline' element should have a 'command' attribute. e.g.
     <inline command="/bin/cat /tmp/a /tmp/b"/>

```

The callout program should return a well-formed XML document. The document's outermost element should be a "substitution" element. It should contain any XML that is valid according to this DTD.



That "substitution" element's contents will replace the "inline" element in the datacapture.cfg file.

So, if this DCT snippet:

```
<select>
<inline command="blah" />
</select>
```

runs the "blah" callout program, and that program returns this text:

```
<substitution>
<option label="ABC" />
<option label="123" />
<option label="Jackson 5" />
</substitution>
```

then the DCT snippet will, after callout execution and inline substitution, look like:

```
<select>
<option label="ABC" />
<option label="123" />
<option label="Jackson 5" />
</select>
-->
```

```
<!ELEMENT inline EMPTY >
  <!ATTLIST inline
    command CDATA #REQUIRED
  >
```

<!-- The rest of these elements are common to both datacapture4.5.dtd and symboltable4.5.dtd. -->

```
<!ELEMENT data-capture-requirements (%data-capture-requirements-
contentspec;) >
  <!ATTLIST data-capture-requirements
    name CDATA #IMPLIED
    type (metadata|content|workflow) #REQUIRED
```

```

        dtd-system-identifier  CDATA                                #IMPLIED
    >

<!ELEMENT ruleset(label?,description?,(%items;)*)>
    <!ATTLIST ruleset
        name      CDATA                #REQUIRED
    >

<!ELEMENT container      (label?,description?,(%items;)* ) >
    <!ATTLIST container
        name          CDATA                #REQUIRED
        hide-name     (t|f)                "f"
        combination   (and|or)             "and"
    >

<!ELEMENT item           (label?,description?,database?(checkbox|radio|
        text|textarea|select|replicant|browser|
        readonly|hidden)+ ) >
    <!ATTLIST item
        name          CDATA                #REQUIRED
    >

<!ELEMENT label          (#PCDATA) >
<!ELEMENT description    (#PCDATA) >

<!ELEMENT readonly      (allowed?,callout?) >
    <!ATTLIST readonly
        rows          CDATA                "0"
        cols          CDATA                "0"
    >

<!ELEMENT hidden        (allowed?,callout?) >
    <!ATTLIST hidden
        required      (t|f)                "f"
    >

<!ELEMENT text          (allowed?,default?,callout?) >
    <!ATTLIST text
        required      (t|f)                "f"
        maxlength     CDATA                "0"

```



```
        size                CDATA                "0"
        validation-regex    CDATA                #IMPLIED
    >
<!-- validation-regex is a Perl regex for validating this element -->

<!ELEMENT textarea        (allowed?,default?,callout?) >
  <!ATTLIST textarea
    required                (t|f)                "f"
    rows                    CDATA                "0"
    cols                    CDATA                "0"
    wrap                    (off|virtual|physical) "off"
    validation-regex        CDATA                #IMPLIED
    rtf                     (t|f)                "f"
  >
<!-- validation-regex is a Perl regex for validating this element -->

<!ELEMENT browser        (allowed?,callout?) >
  <!ATTLIST browser
    required                (t|f)                "f"
    maxlength                CDATA                "0"
    size                    CDATA                "0"
    initial-dir              CDATA                #IMPLIED
    ceiling-dir             CDATA                #IMPLIED
    extns                   CDATA                #IMPLIED
  >

<!ELEMENT checkbox      (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST checkbox
    required                (t|f)                "f"
    delimiter                CDATA                ", "
  >

<!ELEMENT radio         (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST radio
    required                (t|f)                "f"
  >

<!ELEMENT select        (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST select
    required                (t|f)                "f"
    size                    CDATA                "0"
    multiple                (t|f)                "f"
  >
```

```

        delimiter      CDATA          ", "
        width          CDATA          #IMPLIED
    >
<!-- The delimiter attribute is for multiple=t only -->

<!ELEMENT replicant      (allowed?, (%items;)*)>
  <!ATTLIST replicant
    min          CDATA          "0"
    max          CDATA          "1"
    default      CDATA          "1"
    combination  (and|or)      "and"
    hide-name    (t|f)         "t"
  >

<!ELEMENT option        EMPTY>
  <!ATTLIST option
    selected     (t|f)         "f"
    value        CDATA          #IMPLIED
    label        CDATA          #REQUIRED
  >

<!ELEMENT allowed      (cred|and|or|not)>

<!ELEMENT cred         EMPTY>
  <!ATTLIST cred
    role         CDATA          #IMPLIED
    user         CDATA          #IMPLIED
  >

<!ELEMENT and          (cred|and|or|not)+>

<!ELEMENT or           (cred|and|or|not)+>

<!ELEMENT not          (cred|and|or|not)>

<!ELEMENT default      (#PCDATA)>

<!ELEMENT callout      (param*) >
  <!ATTLIST callout

```



```

    type          ( java-class)    #REQUIRED
    label         CDATA             #REQUIRED
    location      CDATA             #REQUIRED
    class         CDATA             #REQUIRED
  >

<!--ELEMENT param          EMPTY -->
  <!--ATTLIST param
    name          CDATA             #REQUIRED
    value         CDATA             #REQUIRED
  >

<!--ELEMENT database      EMPTY -->
  <!--ATTLIST database
    deploy-column ( t|f)           "t"
    searchable   ( t|f)           "t"
    data-type    CDATA             "VARCHAR( 255 )"
    data-format  CDATA             #IMPLIED
  >
```

Setting Up Presentation Templates

Creating Presentation Templates

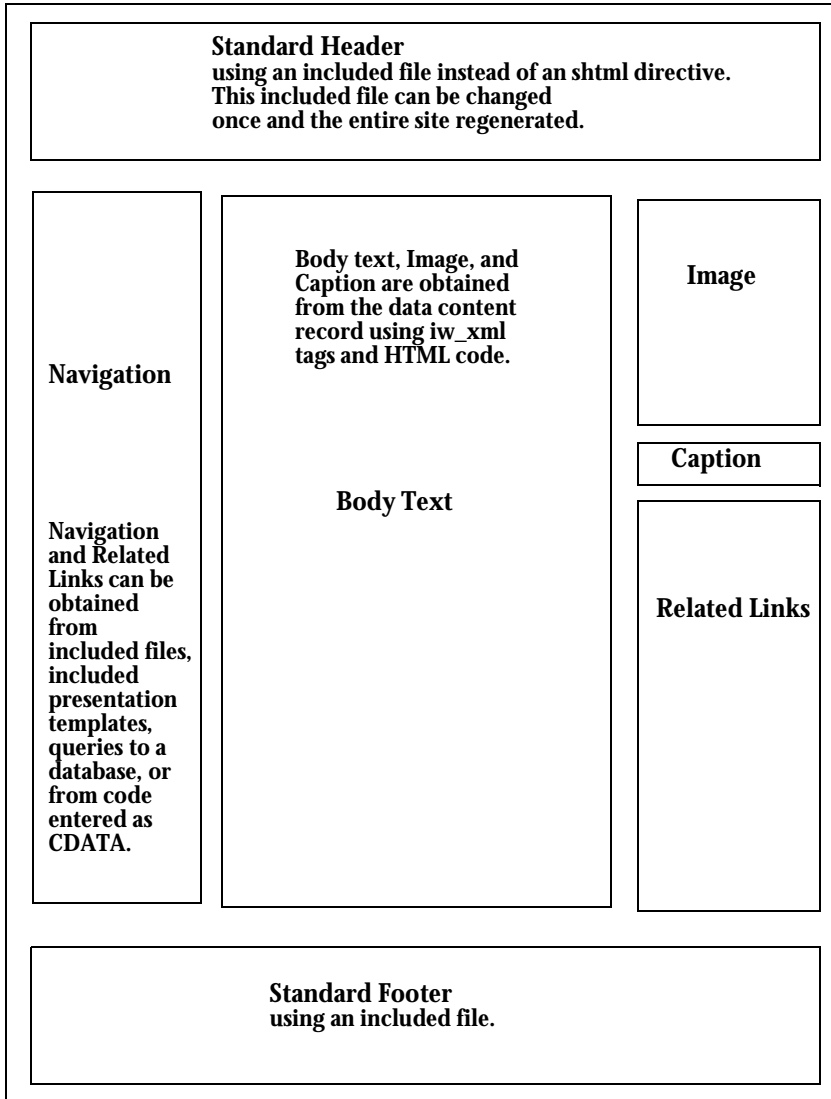
Presentation templates are designed to display data. The data may be obtained from the following sources:

- Data content records
- Queries to relational databases
- Perl-generated output
- Included files
- Included presentation components

You can combine data content records with presentation templates to generate output files. You can also create output files using relational database queries and output generated via the Perl API. TeamXpress Templating can generate any text content, including HTML, XML, or any application server code. Using TeamXpress Templating, you can precompile elements or a dynamic page, maintain dynamic content as application server code, eliminate the need for sever-side includes, and output an `.asp` or `.jsp` file that can be served dynamically at runtime in the production environment. At a minimum, TeamXpress Templating can precompile flat HTML files that can sit as static files to provide maximum performance.

Presentation templates are written in XML and may contain custom Interwoven XML tags, HTML, and Perl.

The following diagram shows a typical output page and describes how the page is generated.



Generating a Web Page with TeamXpress Templating

Presentation templates allow you to:

- Use built-in tags to fetch elements from XML data content records, loop on lists, do SQL queries, perform conditional logic, etc.
- Create custom XML tags that encapsulate arbitrary presentation logic. Non-programmers can use custom high-level visual building blocks without writing any code.
- Create custom libraries and invoke them from within the `<iw_perl>` tag. Lower-level visual building blocks can be accessed by programmers directly from a template.
- Intermix XML and Perl to generate any output format (such as `html`, `asp`, and `jsp`). Presentation information does not need to be hard-coded into the template.
- Make common code components reusable across templates.
- Create generic components (component templates) that display differently based on the parameters they are given by their enclosing template.
- Eliminate page compilation costs on the production web server, thus increasing scalability of your web site.
- Use component templates. The component template may have key, value parameters passed to it by the enclosing template. For example, a component template may include an SQL query whose body depends on parameters from the enclosing template. Component templates do not take a data content record.

To write a presentation template, you must first know some basic XML. Specifically, an understanding of the following XML topics is useful:

- CDATA
- “Well-formed” documents
- Entities (e.g., `>` and `<`)

A useful reference is <http://www.xml.com/axml/testaxml.htm>.

Interwoven XML tags are an important part of writing presentation templates. The following is an overview of the existing tags:

<code><iw_xml></code>	Base class for presentation template XML elements.
<code><iw_pt></code>	Specifies that the document is a presentation template, and names it.
<code><iw_value></code>	Inserts the value of a Perl expression or data content record item.
<code><iw_if></code>	Provides an expression that is evaluated as being either true or false to determine whether the <code><iw_then></code> or <code><iw_else></code> statement will be used.
<code><iw_then></code>	Provides contents to be included if the <code><iw_if></code> tag's expression is true.
<code><iw_else></code>	Provides contents to be included if the <code><iw_if></code> tag's expression is false.
<code><iw_ifcase></code>	Provides for conditional inclusion of contents.
<code><iw_case></code>	Used with <code><iw_ifcase></code> for conditional inclusion of contents.
<code><iw_perl></code>	Executes arbitrary Perl code and provides an API for generating input and using data content records
<code><iw_iterate></code>	Iterates through a data content record or Perl list.
<code><iw_sql_open></code>	Opens a database connection.
<code><iw_sql_iterate></code>	Iterates SQL result sets.
<code><iw_sql_query></code>	Queries a database.
<code><iw_system></code>	Uses output from an external command.
<code><iw_next></code>	Skips to the next iteration of a (possibly labeled) loop.
<code><iw_last></code>	Skips to the last iteration of a (possibly labeled) loop.
<code><iw_include></code>	Inserts a file or the result of compiling a template component in the generated HTML.
<code><iw_repeat></code>	Allows you to repeat content a given number of times.

For more information about the `iw_xml` tags, see “Custom XML Tags” on page 82.

Consider the following guidelines when creating a presentation template:

- When writing presentation templates that obtain information from data content records, refer to the data capture template that the data content records are based on. Make sure that the names of the fields are consistent and that you use `<iw_iterate>` tags in the presentation template (see page 99) if there are `replicant` tags in the data capture template (see Chapter 3, “Setting Up Data Capture Templates”).
- Presentation templates must be well-formed XML. Any HTML contained within a presentation template outside of a CDATA directive must be well-formed in accordance with XML rules.
- The `<iw_value>` tag, unlike all other tags, is interpreted within CDATA sections. If you need to enclose a large body of text (e.g., HTML) with CDATA, you still have access to data values within this region.

Using a Presentation Template—An Example

This section provides an overview to show the use of a presentation template. The section includes an example data content record, a presentation template, and a component template.

The presentation template shows how to use tags to call a component template, include a file, obtain data from a data content record, and iterate through all values of a field in a data content record.

The Press Release presentation template is shown on page 76. In addition to using HTML, it uses many of the `iw_xml` tags. This example is provided as `templatedata/internet/pr/presentation/nested_component_example.tpl` in your TeamXpress Templating installation. This presentation template calls the `simple.tpl` component template (page 80) and accesses a data content record (page 74) to obtain values. The generated press release is shown (page 81).



The data content record that contains the data for the Press Release presentation template follows:

```
<record name="pr2">
  <item delimiter=", " name="Date">
    <value>01.04.2000</value>
  </item>
  <item delimiter=", " name="Headline">
    <value>Interwoven, Inc. Files Registration</value>
  </item>
  <item delimiter=", " name="Secondary Headline">
    <value>Interwoven, Inc., (NASDAQ: IWOV)</value>
  </item>
  <item delimiter=", " name="Introductory Paragraph">
    <value>Interwoven, Inc. is a provider of Web content management
    solutions. Its products are specifically designed to help companies
    rapidly and efficiently build, maintain and extend mission-critical
    Web sites and eBusiness applications. Interwoven's principal
    product, TeamXpress, combines the functions of content management,
    version control, workflow and application development in an open,
    standards-based platform that allows large numbers of contributors
    across an enterprise to add Web content in a well-managed manner.
    </value>
  </item>
  <item delimiter=", " name="Story">
    <value>
      <item delimiter=", " name="Subheading">
        <value>heading1</value>
      </item>
      <item delimiter=", " name="Section Paragraphs">
        <value>
          <item delimiter=", " name="Paragraphs">
            <value>Credit Suisse First Boston will act as the lead
            underwriter of the offering. The co-managers are
            Robertson Stephens; Dain Rauscher Wessels; SoundView
            Technology Group; and Adams, Harkness and Hill, Inc.
            </value>
          </item>
        </value>
      </item>
    </value>
  </item>
</record>
```

```
<value>
  <item delimiter=", " name="Paragraphs">
    <value>A registration statement relating to these
    securities has been filed with the Securities and
    Exchange Commission but has not yet become
    effective. These securities may not be sold nor may
    offers to buy be accepted prior to the time the
    registration statement becomes effective. This press
    release shall not constitute an offer to sell or a
    solicitation of an offer to buy, nor shall there be
    any sale of these securities in any state or
    jurisdiction in which such an offer, solicitation
    or sale would be unlawful prior to registration or
    qualification under the securities laws of any such
    state or jurisdiction.</value>
  </item>
</value>
</item>
</value>
</item>
<item delimiter=", " name="Author">
  <value>ddd</value>
</item>
<item delimiter=", " name="EMail">
  <value></value>
</item>
<item delimiter=", " name="Languages">
  <value>English</value>
</item>
</record>
```



The presentation template for the press release follows:

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<iw_pt name="PressRelease"><![CDATA[
<HTML>
<!-- Begin CDATA Tag -->
<!-- HTML stuff is enclosed in CDATA tag -->

<HEAD>
</HEAD>
<BODY bgcolor="#FFFFFF" link="#0033CC" vlink="#0033CC" alink="0000FF"
TEXT=#000000 BACKGROUND="/templatedata/internet/pr/images/pixel.gif">
  <TABLE WIDTH="720" VALIGN="top" CELLPADDING="0" CELLSPACING="0"
  BORDER="0">
    <TR><TD WIDTH="720">

]]> <!-- End of CDATA Tag -->
<!-- nested component -->
  <iw_include pt='/templatedata/components/simple.tpl'>
    <![CDATA[
      $iw_param{Headline} = iwpt_dcr_value('dcr.Headline');
    ]]>
  </iw_include>

<![CDATA[ </TD></TR><TR><TD><TABLE WIDTH="720"><TR valign="top">
  <TD valign="top"> ]]>

  <iw_include file='templatedata/internet/pr/iwprnavbar.html' />

<![CDATA[ <!-- Begin CDATA Tag -->
  </TD><TD VALIGN="top" WIDTH="510">
]]> <!-- End of CDATA Tag -->

<!-- Begin content area -->

<!-- Headline -->
<P></P>
<br></br>

<!-- Secondary Headline -->
<h3> <iw_value name='dcr.Secondary Headline' /> </h3>

```

Using <iw_pt> to open and name the presentation template; beginning CDATA

Using the <iw_include> tag to call the simple.tpl component template

Passing "Headline" value as a parameter to component template

Using the <iw_include> tag to include an HTML file

```
<!-- Date -->  
<P> <B>SUNNYVALE, Calif., <iw_value name='dcr.Date' />:</B></P>
```

```
<!-- Introductory Paragraph -->  
<P><iw_value name='dcr.Introductory Paragraph' /> </P>
```

```
<!-- Story -->  
<iw_iterate var='story' list='dcr.Story'>
```

```
  <!-- Subheading -->  
  <iw_value name='story.Subheading' />
```

```
    <!-- Paragraphs -->  
    <iw_iterate var='para_value' list='story.Section Paragraphs'>  
      <p><iw_value name='para_value.Paragraphs' /></p>  
    </iw_iterate>
```

```
</iw_iterate>
```

```
<!-- Insert 'aboutIW.html' file -->  
<p>  
<iw_include file='templatedata/internet/pr/aboutIW.html' />  
</p>
```

```
<p>For more information on the company and  
its software solutions, visit the Interwoven Web site at  
<a href="http://www.interwoven.com">www.interwoven.com</a>  
or e-mail <a href="mailto:{iw_value name='dcr.EMail'}">  
  <iw_value name='dcr.EMail' /></a>  
</p>
```

```
<!-- HTML stuff is enclosed in CDATA tag -->  
<![CDATA[  <!-- Begin CDATA tag -->
```

```
<p>  
<TABLE WIDTH=520 BORDER=0 CELSPACING=10 CELLPADDING=0>  
<TR>  
  <TD COLSPAN=2 BGCOLOR=#999999>  
    <IMG SRC="/templatedata/internet/pr/images/pixel.gif">  
  </TD>  
</TR>
```

Obtaining a value from a data content record using the <iw_value> tag

Nesting of <iw_iterate> tags

Using <iw_iterate> and <iw_value> to obtain multiple paragraph values

Opening CDATA containing HTML

```

<TR>
  <TD COLSPAN=2>    <BR>
    <!-- Begin question -->
    <CENTER>
    <FONT SIZE=3><B>
    <A HREF="/customers/profiles/bestbuy.html">How is Best Buy pushing
    billions of dollars in business towards the Web?</A>
    </B></FONT>
    </CENTER>
    <!-- End question -->
  </TD>
</TR>
<TR>
  <TD COLSPAN=2>&nbsp;    
  </TD>
</TR>
<TR>
  <TD COLSPAN=2 BGCOLOR=#999999>
    <IMG SRC="/templatedata/internet/pr/images/pixel.gif">
  </TD>
</TR>
<TR>
  <TD WIDTH=250 VALIGN="top">
    <IMG SRC="/templatedata/internet/pr/images/pixel.gif"
    WIDTH=250 HEIGHT=1>
    <FONT SIZE="1" COLOR=#666666> <B>Global Headquarters</B><BR>
    Interwoven, Inc.<BR>    1195 W. Fremont Ave. #2000<BR>
    Sunnyvale, CA 94087 US<BR>    Phone: (408) 774-2000<BR>
    </FONT>
  </TD>
  <TD WIDTH="250" VALIGN="top"><BR><FONT SIZE=1 COLOR=#666666>
    <B>How Can We Serve You?</B><BR>
    Let us know at:<A HREF="mailto:info@interwoven.com">
    info@interwoven.com</A><BR> or <A HREF="/cgi-bin/reg">Register</A>
    and we will contact you!<BR>
    <BR><NOBR>Web Team: <A HREF="mailto:webteam@interwoven.com">
    webteam@interwoven.com</A>
    </NOBR></FONT>
  </TD>
</TR>

```



```
<TR>
  <TD COLSPAN=2 ALIGN=CENTER>    <BR>
    <FONT COLOR=#666666>    <P CLASS="copyright">
      <A HREF="/copyright.html">    Copyright &copy; 2000</A>
      Interwoven, Inc. All rights reserved. </P> </FONT>
    </TD>
  </TR>
</TABLE>
</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
]]>  <!-- End CDATA tag -->
</iw_pt>
```

Closing CDATA
containing HTML

Closing the
presentation template



The `simple.tpl` component template is called from the main presentation template. It prints the headline it obtains from the calling presentation template. The contents of the `simple.tpl` component template is:

```
<?xml version="1.0" standalone="yes"?>
<iw_pt name="Banner Component PT">
<!-- This is a component PT that can be used inside another PTs -->
<!-- It prints the title that it got from the container PT -->
<TABLE width="720" align="center">
  <TR align="center">
    <TD align="left">
      <IMG SRC="/templatedata/internet/pr/images/iw-logo-small.gif"
        WIDTH="220" HEIGHT="40" BORDER="0"/>
    </TD>
    <TD align="center">
      <H1><iw_value name='$iw_arg{Headline}' /></H1>
    </TD>
  </TR>
</TABLE>
</iw_pt>
```

Tag that opens
and names the
component template

Tag that obtains the
headline from the
calling template

The press release generated from this presentation template, component template, and data content record would be as follows:

INTERWOVEN

home
point of view
products
customers
partners
company
investors
news
press
coverage
awards
events
services
library
careers
feedback

Interwoven, Inc. Files Registration

Interwoven, Inc., (NASDAQ: IWOV)
SUNNYVALE, Calif., 01.04.2000:

Interwoven, Inc. is a provider of Web content management solutions. Its products are specifically designed to help companies rapidly and efficiently build, maintain and extend mission-critical Web sites and eBusiness applications. Interwoven's principal product, *TeamXpress*, combines the functions of content management, version control, workflow and application development in an open, standards-based platform that allows large numbers of contributors across an enterprise to add Web content in a well-managed manner.

heading 1

Credit Suisse First Boston will act as the lead underwriter of the offering. The co-managers are Robertson Stephens; Dain Rauscher Wessels; SoundView Technology Group; and Adams, Harkness and Hill, Inc.

A registration statement relating to these securities has been filed with the Securities and Exchange Commission but has not yet become effective. These securities may not be sold nor may offers to buy be accepted prior to the time the registration statement becomes effective. This press release shall not constitute an offer to sell or a solicitation of an offer to buy, nor shall there be any sale of these securities in any state or jurisdiction in which such an offer, solicitation or sale would be unlawful prior to registration or qualification under the securities laws of any such state or jurisdiction.

For more information on the company and its software solutions, visit the Interwoven Web site at www.interwoven.com or e-mail

How is Best Buy pushing billions of dollars in business towards the Web?

Global Headquarters
Interwoven, Inc.
1195 W. Fremont Ave. #2000
Sunnyvale, CA 94087 US
Phone: (408) 774-2000

How Can We Serve You?
Let us know at: inb@interwoven.com
or Register and we will contact you!

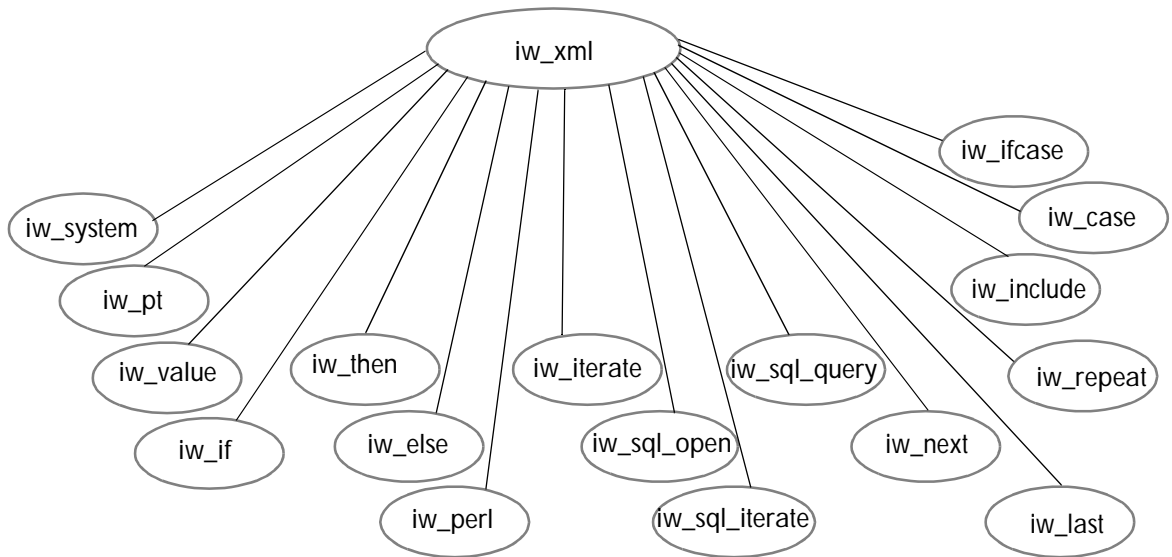
Web Team: webteam@interwoven.com

Copyright © 1999 Interwoven, Inc. All rights reserved.

Local machine zone

Custom XML Tags

A number of custom XML tags are supplied with TeamXpress Templating. All of these tags are derived from a base class, `iw_xml`, as shown below.



The `iw_xml` tags are described in this section. Some of the examples are taken from the Yacht Info and Press Release data capture forms described in the preceding chapter.

Typical man pages are available online for each tag. If `iw-home/iw-perl/bin` is in your path statement, you can access these man pages by issuing the command `perldoc TeamSite::PT::tag_name`.

<iw_case>

The <iw_case> tag is used to perform boolean tests (such as equality, inequality or regex match) on the value, attribute value, and/or type of the variable specified in the `name` attribute of the enclosing <iw_ifcase> tag (page 91).

The content of only one <iw_case> within an <iw_ifcase> is selected to form the final output page. If the boolean test of more than one <iw_case> tag is true, the first true <iw_case> section is selected (Example 1).

If an <iw_case> tag has no attributes, it becomes the default case of its enclosing <iw_ifcase> tag. You may not use more than one default <iw_case> within the enclosing <iw_ifcase>. If a default <iw_case> is present, it must be the final <iw_case> of the enclosing <iw_ifcase>.

Attributes

The <iw_case> tag has the following six optional attributes: `value`, `attrib`, `type`, `op`, `expr`, and `expression`.

<code>value</code>	May be used with <code>op</code> and <code>attrib</code> . If the variable named in <iw_ifcase> is an XML node, <code>value</code> corresponds to the CDATA of that XML node. If <iw_ifcase> names a non-XML node Perl variable, then <code>value</code> refers to the variable's value (Example 2).
<code>attrib</code>	Must be used with <code>value</code> ; may be used with <code>op</code> . The statement <iw_case attrib='x' value='y'> is true if the XML node being tested by the enclosing <iw_ifcase> has an attribute named <code>x</code> with a value of <code>y</code> .
<code>type</code>	May be used with <code>op</code> . When iterating over all XML nodes of all types within <code>y</code> , use <code>type='...'</code> in an <iw_case> tag to discover the current node's type.

You may use any combination of the following within the <iw_case> tags of the same <iw_ifcase> statement (the first true <iw_case> within a given <iw_ifcase> statement will be selected):

```

value      = '...'
type       = '...'
expr       = '...'
expression = '...'
attrib     = '...'      value='...'
    
```

`op` May be used with `value` and `type`. When testing `value='...'` or `type='...'` in an `<iw_case>` tag, `eq` (string equality) is the default `op` attribute.

`op` may have the values indicated in the following table:

Value	Definition
<code>eq</code>	equal to
<code>ne</code>	not equal to
<code>lt</code>	less than
<code>gt</code>	greater than
<code>le</code>	less than or equal to
<code>ge</code>	greater than or equal to
<code>=~</code>	regular expression match
<code>!~</code>	regular expression does not match

Therefore, `op='=~' value='red'` means “regex match on the pattern `red`”. In other words, the `<iw_ifcase>` value contains the substring `red`.

`expr` The `expr` attribute is a more powerful version of the `value` attribute (but it is less powerful than `expression`).

In `expr`, the value of the `<iw_ifcase>` is the implicit left-hand side of the general Perl expression that you provide. This allows you to use regex patterns drawn from variables, regex switches, etc.

The `expression` attribute of `<iw_case>` is the completely general conditional test; nothing is implicit.

Therefore, the following combinations are equivalent (they match values equal to the string `red`):

```
<iw_case value = 'red'> ...
<iw_case op = 'eq' value = 'red'> ...
<iw_case expr = "eq 'red'"> ...
<iw_case expression = " $iw_ifcase_value eq 'red'">
...

```

The following combinations are also equivalent (they match values that *start* with the string `red`).

```
<iw_case op = '=~' value='^red'> ...
<iw_case expr = '=~ /^red/'> ...
<iw_case expression = ' $iw_ifcase_value =~ /^red/'> ...

```

Usually, you will only need to use `value='...'` or `type='...'`

Note: the `i` switch means “case insensitive matching” as in the following example:

```
<iw_case expr='=~ /goldfish/i' >
```

`expression`

The `<iw_ifcase>` tag uses its `name='...'` attribute to set three variables that enclosed tags can inspect:

```
$iw_ifcase_type (type)
$iw_ifcase_value (value)
$iw_ifcase_attr (attribute hash reference)
```

Typically, only `iw_case` tags use these variables, but any tag may do so. This allows `iw_ifcase` to evaluate `name='...'` only once regardless of how many `<iw_case>` tags are inside it.

When one `<iw_ifcase>` tag is nested within another, the inner `$iw_ifcase_...` variables hide the `$iw_ifcase_...` variables of the outer tag. In fact, you can choose non-default names to avoid this.



See the example section of `<iw_ifcase>` (page 91) for more information.

The following code shows switching on an arbitrary function, `length` (this function is a standard/built-in Perl function):

```
<iw_case expression=' length($iw_ifcase_value) >
15'>
```

Example 1

In the following example, either the first or second `<iw_case>` statement will be selected, depending on the information contained in CDATA.

```
<iw_ifcase name='dcr.customer.lastname'>
  <iw_case value='Smith'>
    If the CDATA portion of the "lastname" element within
    the top-level "customer" tag is "Smith", then this is
    the case that will be selected.

  </iw_case>
  <iw_case type='lastname'>
    This test will always be true since the enclosing
    iw_ifcase is explicitly testing an XML node of type
    "lastname"; however, if the customer's name is "Smith"
    the initial iw_case will be selected since it appears
    first (thus the iw_case chain has if-else-if-else...
    semantics).
  </iw_case>
  <iw_case value='Jones'>
    This case will never be reached: even if
    the first case is false, the second case
    (type='lastname') will always be true.
  </iw_case>
</iw_ifcase>
```


Example 2

The following example tests the 0th CDATA section within the `egg` XML node to check if it is equal to the string `Grade A` (unless the `attrib` attribute is also used; see `attrib` on page 83).

```
<iw_ifcase name='dcr.hen.egg'>
  <iw_case value='Grade A'> ...
</iw_case>
</iw_ifcase>
```

Example 3

```
<iw_iterate var='node_within_y' list='dcr.x.y.*'>
  <iw_ifcase name='node_within_y'>
    <iw_case type='customer'>
      do this if the node is of type customer
    </iw_case>
    ...
  </iw_ifcase>
</iw_iterate>
```

Refer to `iw-home/examples/Templating/templatedata/tutorials`.

<iw_else>

The <iw_else> tag is used for conditional inclusion of contents. It is used with the <iw_if> tag and provides contents to be included if the <iw_if> tag is false. The <iw_then> tag (page 121) is also used and provides the contents if the <iw_if> tag (page 89) is true.

For if-else-if... conditional statements, you may also use <iw_ifcase> (page 91) and <iw_case> (page 83).

Examples

The following examples take this form:

```
<iw_if expr=' some logical condition ' >
  <!-- optional then clause (only included when iw_if is true) -->
  <iw_then>...</iw_then>
  <!-- optional else clause (only included when iw_if is false) -->
  <iw_else>...</iw_else>
</iw_if>
```

Example 1

```
<iw_if expr=' {iw_value name="$iw_arg{moo}"/} eq "cow" ' >
  <iw_then>
    do this if the condition is true
  </iw_then>
  <iw_else>
    do this if the condition is false
  </iw_else>
</iw_if>
```

Example 2

```
<iw_if expr=' ( {iw_value name="dcr.xyz"} > 42 ) ||
              ( {iw_value name="$iw_arg{pdq}"/} < 99 )
              '>
  <iw_then>
    do this if the condition is true
  </iw_then>
</iw_if>
```

<iw_if>

The <iw_if> tag is used with an expression that is evaluated to determine whether the <iw_then> (page 121) or <iw_else> (page 88) statement will be used.

For if-else-if... conditional statements, you may also use <iw_ifcase> (page 91) and <iw_case> (page 83).

Attributes

The <iw_if> tag requires the *expr* attribute.

expr = 'expression' A Perl logical expression, in XML-encoded form (for example, '<' must be encoded as '<' and '&' must be represented as '&').

If the expression evaluates to true, the enclosed <iw_then> clause is used when generating the presentation template's output. Otherwise, the optional enclosed <iw_else> clause is used.

expr may contain one or more instances of {iw_value name='...'/} within the logical expression. This allows the <iw_if> tag to branch conditionally on values within the data content record and/or Perl variables.

The semantics of {iw_value name='...'/} within the *expr* attribute are identical to the <iw_value> tag (page 122).

Note: Because the character '<' is not permitted in an attribute list, XML does not accept the statement:

```
<iw_if expr="<iw_value name='$moo'/"> eq 'cow' ">
```

However, when you use the '{' character, the statement is accepted:

```
<iw_if expr="{iw_value name='$moo'/"> eq 'cow' ">
```

For EBNF details, see <http://www.xml.com/axml/testaxml.htm> (search for the second occurrence of 'AttValue').

Examples

All three of the following examples take this form:

```
<iw_if expr=' some logical condition ' >
  <!-- optional then clause (only used when iw_if is true) -->
  <iw_then>...</iw_else>
</iw_if>
```

Example 1

```
<iw_if expr=' {iw_value name="$iw_arg{moo}"/} eq "cow" ' >
  <iw_then>
    do this if the condition is true
  </iw_then>
  <iw_else>
    do this if the condition is false
  </iw_else>
</iw_if>
```

Example 2

```
<iw_if expr=' {iw_value name="dcr.price"/} > 42 ' >
  <iw_then>
    do this if the condition is true
  </iw_then>
</iw_if>
```

Example 3

```
<iw_if expr='0'>
  <iw_then>
    this statement is now commented out!
  </iw_then>
</iw_if>
```

<iw_ifcase>

The <iw_ifcase> tag is used for conditional inclusion of contents. The <iw_ifcase> tag uses its immediately enclosed <iw_case> (page 83) tags to form a single if-else-if-else... chain.

Attributes

The <iw_ifcase> tag has one required attribute, `name`, and three optional attributes, `iw_ifcase_value`, `iw_ifcase_type`, and `iw_ifcase_attrib`.

`name` Specifies either a Perl variable or an XML node in the data content record. The semantics are identical to the `name` attribute in <iw_value> (page 122):

name	corresponds to
dcr.x.y	y component of top-level DCR/XML node x
hen.egg	the egg XML node within the hen node
cow@mo0	mo0 attribute of DCR/XML node cow
z	the Perl variable \$z
\$z	the Perl variable \$z
(....)	the result of the Perl expression

The immediately enclosed <iw_case> tags use the value corresponding to `name` to test for conditional inclusion of contents; the contents of the first <iw_case> tag with a logical test that returns boolean true are included.

If `name` corresponds to an XML node in the DCR (as opposed to something like a Perl string variable), the immediately enclosed <iw_case> tags may also use the type and attribute list values of this XML node to determine which <iw_case> is used.

See <iw_case> (page 83) for more details.

`iw_ifcase_value` See Advanced Usage.

`iw_ifcase_type` See Advanced Usage.

`iw_ifcase_attr` See Advanced Usage.

Advanced Usage

The `<iw_ifcase>` tag normally sets three variables that are used by enclosed `<iw_case>` statements.

Usually, they are named:

1. `$iw_ifcase_value` (the value of `name='...'`)
2. `$iw_ifcase_type` (the type of `name='...'`)
3. `$iw_ifcase_attr` (attribute hash of `name='...'`)

In rare cases, nested `<iw_ifcase>` tags may require inner `<iw_case>` tags to access the value/type/attributes of both the inner and outer `<iw_ifcase>`. These optional attributes allow variables (1), (2), and (3) to have names other than `$iw_ifcase_...`

This allows for code like this:

```
<iw_ifcase name='x' iw_ifcase_type='x_type'>
  <iw_case type='animal'>
    <!-- x is of type animal -->
    <iw_ifcase name='y' iw_ifcase_type='y_type'>
      <iw_case type='vegetable'>
        <!-- y is of type vegetable -->
        The type of x is: <iw_value name='x_type' />
        The type of y is: <iw_value name='y_type' />
      </iw_case>
    </iw_ifcase>
  </iw_case>
</iw_ifcase>
```

Example 1

```

<iw_ifcase name='dcr.x.y'>
  <iw_case value='red'>
    The XML node accessed via dcr.x.y has the value "red"
  </iw_case>

  <iw_case op='=~' value='green'>
    The XML node accessed via dcr.x.y includes the string "green"
  </iw_case>

  <iw_case attrib='shoesize' value='10'>
    The XML node accessed via dcr.x.y has a
    "shoesize" attribute with a value of "10"
  </iw_case>

  <iw_case>
    This case handles everything else
  </iw_case>
</iw_ifcase>

```

Example 2

When iterating over every XML node of all types within *y*, you can use `type='...'` in the `<iw_case>` tag to discover the current node's type.

```

<iw_iterate var='an_xml_node_within_y' list='dcr.x.y.*'>
  <iw_ifcase name='an_xml_node_within_y'>
    <iw_case type='customer'>
      This is an XML element of type "customer"
      See for yourself: <iw_value name='$iw_ifcase_type' />
    </iw_case>

    <iw_case op='=~' type='^zzz_'>
      The type of this XML element starts with "zzz_"
    </iw_case>

    <iw_case value='zebra'>
      Any XML node type whose value is equal to "zebra"
    </iw_case>
  <iw_case type='animal'>

```



```
    The type of this XML node is "animal"  
    non-zebra animal: <iw_value name='an_xml_node_within_y' />  
</iw_case>  
<iw_case>  
    The default case  
</iw_case>  
</iw_ifcase>  
</iw_iterate>
```


<iw_include>

The <iw_include> tag allows you to insert a file or component presentation template at the point where this tag appears.

Attributes

The <iw_include> tag requires one of the two attributes, `file` or `pt`, and allows two optional attributes, `ienc` and `mode`.

<code>file='filepath'</code>	Specifies the path to the file to include. The contents of the file are included where the <iw_include> tag occurs.
<code>ienc='encoding'</code>	Specifies encoding when a file named by the <code>file</code> attribute is not encoded in UTF-8. Mandatory when a file specified by the <code>file</code> attribute does not use UTF-8 encoding.
<code>pt='filepath'</code>	Specifies the path to the presentation template to process and include at the point where the <iw_include> tag occurs. You may nest any number of templates, although performance may degrade if nesting is excessive.
<code>mode='mode'</code>	Specifies whether <code>filepath</code> is a relative or absolute path (Example 1).

`mode` has three possible values:

- `docroot` (default value): The path specified on the command line (via `-iw_include-location path`) is always prepended to the file name given in the `file` or `pt` attribute. See “`iwpt_compile.ipl`” on page 372. From the TeamXpress GUI, it is as if the `docroot` of the file system is the base of the user’s workarea.
- `ptlocal`: Relative path names are relative to the directory of the current presentation template. However, absolute paths are absolute in relation to the computer’s file system.
- `cwd` (current working directory). Relative path names are relative to the path specified by the command line (via `-iw_include-location path`). However, absolute paths are absolute in relation to the computer’s file system.

A file path is relative only under the following conditions:

- On Unix: file name does not begin with a slash.
- On Windows: file name does not begin with a *<driveletter>*.

When `docroot` mode is used (or when `cwd` mode is used with a relative path), the `-iw_include-location` flag must be specified with `iwpt_compile`.

The following table shows how file paths change given different modes and path types (relative versus absolute):

Mode	Given relative path: daughter/son.html	Given absolute path: /etc/passwd
docroot (default)	<i>path</i> /daughter/son.html	<i>path</i> /etc/passwd
ptlocal	<i>ptpath</i> /daughter/son.html	/etc/passwd
cwd	<i>path</i> /daughter/son.html	/etc/passwd

ptpath is the path of the current presentation template (or template component).

CDATA

CDATA is an optional tag that is only meaningful in `pt`-style inclusion.

CDATA may contain Perl code that initializes a hash named `%iw_param`. The `%iw_param` hash defines the key, value pairs that are available within the nested template via the `%iw_arg` hash (Example 2).

Example 1

```
<iw_include file='exampleFile' mode='docroot' />
```

The literal contents of the file `exampleFile` will be inserted at the point where this tag appears. The file path can be relative or absolute, based on the `mode` attribute. Since `mode` is `docroot` (and if the path starts with a slash or a drive letter), the file path given on the command line (using the `-iw_include-location path` flag) will be prepended to the file name given in the `file` or `pt` attribute of `iw_include`.

Example 2

If the enclosing template is:

```
<iw_include pt='hello_nested.tpl'><![CDATA[
  $iw_param{color}      = "green";
  $iw_param{moo}        = "cow";
  $iw_param{headline} = iwpt_dcr_value('dcr.Headline');
  ]]>
</iw_include>
```

then within the enclosed template, the line:

```
Your favorite color is <iw_value name='$iw_arg{color}'/>!
```

produces the output:

```
Your favorite color is green!
```

Synopsis

Various usage examples follow:

```
<iw_include file ='myfile.html' />

<iw_include file ='myfile.html'
           ienc ='GB2312' />

<iw_include file =' /shared/stuff/a_very_large_table.html'
           mode ='cwd' />
  <iw_include pt   ='myfile.tpl' />

<iw_include pt   ='myfile.tpl'
           mode ='ptlocal' />
```



```
<iw_include pt    = 'hello_nested.tpl'  
            mode = 'ptlocal' />  
  <![CDATA[  
    $iw_param{color}    = "red";  
    $iw_param{moo}      = "cow";  
    $iw_param{headline} = iwpt_dcr_value('dcr.Headline');  
  ]]>  
</iw_include>
```

<iw_iterate>

The <iw_iterate> tag is used to iterate over a data content record (DCR) list or a Perl list. The <iw_iterate> tag can be nested to any number of levels; that is, there can be other <iw_iterate> tags inside an <iw_iterate> tag.

Attributes

The <iw_iterate> tag has two required attributes, `list` and `var`, and three optional attributes, `order`, `iteration`, and `label`.

- | | |
|---------------------------------------|---|
| <code>var='variable_name'</code> | Defines the iterator variable that contains the current value within the list. <code>variable_name</code> cannot include the space character. the <code>var</code> attribute is transformed into a Perl variable of the same name, which is accessible via <iw_value> and within <iw_perl> code. |
| <code>list='list'</code> | Specifies the list being iterated over. If the <code>list</code> attribute's value is not enclosed in parentheses, it is assumed to be referring to a DCR list (or a <code>var</code> -declared iterator into a DCR list). If the <code>list</code> attribute's value is enclosed in parentheses, then it is a Perl expression that evaluates to a list. Perl-based lists may be created manually within an <iw_perl> tag or inserted into the presentation template automatically by other tags that make use of this feature. |
| <code>order='sort_order'</code> | Specifies a function that reorders the list. |
| <code>iteration='counter_name'</code> | Overrides the name of the counter used by tags, such as <iw_if>, to determine how many elements within a list have been iterated through. The default name of the iteration counter is <code>iw_iteration</code> . |
| <code>label='label'</code> | Allows <iw_next> (page 105) and <iw_last> (page 104) to exit deeply nested looping structures. |



Example 1

The following example uses the iteration attribute with `<iw_if>` and `<iw_then>` tags.

```
<iw_iterate var      = 'current_color'
           list      = '@colors'
           iteration = 'color_count'>

  <iw_iterate var      = 'current_shape'
           list      = '@shapes'
           iteration = 'shape_count'>

    <iw_if expr='!(((iw_value name="$color_count"/} +
                    {iw_value name="$shape_count"/})) %10) '>

      <iw_then>
        do some special magic every
        0th, 10th 20th (..etc) time around the loop!
      </iw_then>
    </iw_if>

    Here's a color: <iw_value name='current_color' />
    Here's a shape: <iw_value name='current_shape' />
  </iw_iterate>
</iw_iterate>
```

Example 2

The following example generates Perl variables named `"$Heading"` and `"$Sect"`.

```
<iw_iterate var='Sect' list='dcr.Section'>
  <iw_iterate var='Heading' list='Sect.heading'>
    <iw_value name='Heading' />
  </iw_iterate>
</iw_iterate>
```

Example 3

The following example shows lists created manually with an `<iw_perl>` tag.

```
<iw_perl><![CDATA[    my %h=(k1=>'v1', k2=>'v2');    ]]>
</iw_perl>

<iw_iterate var='$kname' list='(keys %h)'\>
  <iw_value name='$kname' />    (a key name in hash %h)
</iw_iterate>
```

Example 4

The following example iterates in reverse order:

```
<iw_iterate var = 'x'
          order = 'reverse'
          list = 'dcr.data.customer'\>
  <iw_value name='x' />
</iw_iterate>
```

Example 5

The following example iterates using a custom sort function (in this example, suppose each customer has a `lastname` tag enclosed within it):

```
<iw_perl><![CDATA[
  sub your_function
  {
    # Here's an arbitrary function:
    #
    #   sort customers on the basis of
    #   the length of their last name

    length(iwpt_dcr_value('b.lastname')) <=>
    length(iwpt_dcr_value('a.lastname'));
  }
]]></iw_perl>

<iw_iterate var    = 'x'
```

```
        order = 'sort your_function'
        list  = 'dcr.data.customer'>
    <iw_value name='x' />
</iw_iterate>
```

Synopsis

```
<iw_iterate var='iter' list='(@an_array)'> ...
</iw_iterate>
```

```
<iw_iterate var='iter' order='reverse' list='(@an_array)'> ...
</iw_iterate>
```

```
<iw_iterate var='iter' order='sort your_sort_function'
list='(@an_array)'>...
</iw_iterate>
```

Advanced:

You can also filter the list. The `<iw_iterate>` tag will iterate over whatever *your_arbitrary_function* returns.

```
<iw_iterate var='iter' order='your_arbitrary_function'
list='(@an_array)'>...
</iw_iterate>
```

```
<iw_iterate var='iter' list='(@an_array)' label='xyz'> ...
</iw_iterate>
```

```
<iw_iterate var='iter' list='(jon())'> ...
</iw_iterate>
```

```
<iw_iterate var='iter' list='(keys %h)' iteration='iter_hkey'> ...
</iw_iterate>
```

```
<iw_iterate var='Sect' list='dcr.Section'> ...
</iw_iterate>
```

```
<iw_iterate var='Sect' list='dcr.Section' iteration='iter_sect'> ...
</iw_iterate>
```



```
<iw_iterate var='iter' list='( expr that returns a perl list )'> ...
</iw_iterate>
```

If you specify a custom sorting function via the `order` attribute, (for example, `compare_last_name`), you can define it in an `<iw_perl>` tag:

```
<iw_perl><![CDATA[
  sub compare_last_name
  {
    iwpt_dcr_value('a.last_name') cmp
    iwpt_dcr_value('b.last_name');
  }
]]>
</iw_perl>
```

```
<iw_iterate var    ='cust'
  order  ='sort compare_last_name'
  list   ='dcr.customer'>

  <iw_value name='cust.first_name' />
  <iw_value name='cust.last_name' />
</iw_iterate>
```

<iw_last>

The <iw_last> tag is used to skip to the last iteration of a (possibly labeled) loop.

Attributes

The <iw_last> tag has one optional attribute, `label`.

<code>label='label'</code>	Allows the flow of control to jump outside of the current loop tag (for example, <iw_iterate>) if no explicit label is given. If a label is given that corresponds to a loop label set up in <iw_iterate> or elsewhere, then the flow of control passes to immediately outside that labeled loop. This is useful if you must jump out of a deeply nested looping structure.
----------------------------	---

Example 1

```
<iw_iterate var='Sect' list='dcr.Section' iteration='iter_sect'
  <iw_if expr=' {iw_value name="iter_sect"/} == 3 ' >
    <iw_then>
      <iw_last/>
      <!-- exit the current iw_iterate loop -->
    </iw_then>
  </iw_if>
</iw_iterate>
```

Example 2

```
<iw_iterate var          ='some_xxx_element'
  list              ='(@xxx)'
  iteration         ='nrow'
  label             ='moo'>
  <iw_iterate var='some_yyy_element' list='(@yyy)' >
    <iw_if expr=' {iw_value name="nrow"/} == 3 ' >
      <iw_then>
        <iw_last label='moo' />      <!-- deeply nested! -->
      </iw_then>
    </iw_if>
    ...do some arbitrary stuff...
  </iw_iterate>
</iw_iterate>      <!-- you can directly jump here -->
```

<iw_next>

The <iw_next> tag is used for skipping to the next iteration of a (possibly labeled) loop. It allows the flow of control to skip to the next element of the current loop tag (for example, <iw_iterate>) if no explicit label is given.

Attributes

The <iw_next> tag has one optional attribute, `label`.

<code>label='label'</code>	Allows the flow of control to jump outside of the current loop tag (for example, <iw_iterate>) if no explicit label is given. If a label is given that corresponds to a loop label set up in <iw_iterate> or elsewhere, then the flow of control passes to immediately outside that labeled loop. This is useful if you must exit a deeply nested looping structure.
----------------------------	--

Example 1

```
<iw_iterate var='Sect' list='dcr.Section' iteration='iter_sect'>
  <iw_if expr=' {iw_value name="iter_sect"/} == 3 ' >
    <iw_then>
      <iw_next/>    <!-- skip item 3 -->
    </iw_then>
  </iw_if>
</iw_iterate>
```

Example 2

```
<iw_sql_iterate result_set = 'hulas'
                var        = 'current_dance'
                iteration  = 'nrow'
                label      = 'hula_loop'>
  <iw_if expr=' {iw_value name="nrow"/} == 3 ' >
    <iw_then>
      <!-- skip item 3 (using optional explicit loop name!) -->
      <iw_next label='hula_loop' />
    </iw_then>
  </iw_if>
</iw_sql_iterate>
```

<iw_perl>

The <iw_perl> tag can be used to insert arbitrary Perl code into the presentation template. It has no attributes. The actual Perl code must be inside a CDATA section.

- Arbitrary strings may be created as a part of the presentation template's output. For example:

```
my $first_name = 'Jon';
iwpt_output("Hi there $first_name");
```

- Scalars within a data content record may be accessed. For example:

```
my $headline = iwpt_dcr_value('dcr.Headline');
```

- Lists within a data content record may be traversed/accessed. For example:

```
foreach my $para_iter
  (iwpt_dcr_list('dcr.Story.Section Paragraphs[1].Paragraphs'))
{
  my $paragraph = iwpt_dcr_value('para_iter') ;
  iwpt_output( "here is a paragraph:\n$paragraph\n");
}
```

- Any scalar variable created within an <iw_perl> tag is accessible via the <iw_value> tag. See <iw_value> for details.
- The code created by the <iw_perl> tags and the other tag types forms a single program. This program is what generates the output of `iwpt_compile.ipl`, which is typically an HTML file (see Appendix B, “Using Command-Line Tools”).

Therefore, if you set a variable in one <iw_perl> section, it is available in subsequent <iw_perl> tags as long as it is in scope. If you define a subroutine anywhere, it is accessible everywhere.

APIs

- `iwpt_output ($string)`
`iwpt_output ($string,$ienc)`

Outputs *\$string* to the generated page. If the optional *\$ienc* parameter is given, you may specify the current encoding scheme used by *\$string*, thus allowing it to be properly UTF-8 normalized (UTF-8 normalization must occur even if the final output is not UTF-8).

- `iwpt_dcr_value($accessor_string)`

Fetches the value of a DCR node. If no such node exists, `undef` is returned.

- `iwpt_dcr_list($accessor_string)`
Fetches a list of DCR nodes.
- `iwpt_get_flag_param($flag)`
Gets the list of values associated with an `iwpt_compile.ipl` command-line flag.
- `iwpt_get_ofile_name()`
`iwpt_get_ofile_name($part)`
Gets the name of the file that `iwpt_compile.ipl` will output.
If `$part` is undefined, the entire file name is returned.
If `$part` is 'dirname', the directory portion of the name is returned.
If `$part` is 'basename', the file name (minus the `dirname`) is returned.
- `iwpt_get_dcr_name()`
`iwpt_get_dcr_name($part)`
Gets the name of the DCR used by `iwpt_compile.ipl`.
- `iwpt_get_pt_name()`
`iwpt_get_pt_name($part)`
Gets the name of the presentation template used by `iwpt_compile.ipl`.

See corresponding `TeamSite::PT::iw_xml` docs for: `get_ofile_name`, `get_flag_param`, `get_dcr_name`, `get_pt_name`.

Example

```
<iw_pt>
  ... whatever...

  <iw_perl><![CDATA[
    # access a DCR value
    my $headline = iwpt_dcr_value('dcr.Headline');
    # my $headline = iwpt_dcr_value('dcr.Headline[0]');
    # my $headline = iwpt_dcr_value('dcr.Headline[1]');

    # Let's perform what could be an arbitrarily
    # complex manipulation of a value. The next line
    # of code makes our headline all upper-case!
```



```

$headline = uc $headline;

# now emit it!
iwpt_output( " <h3>Manipulated content:</h3>
              <h2>    $headline    </h2>
              "
              );

# We can do anything at all in here, and then
# manually push our "output" into the page produced
# by the presentation-template engine. In fact, our
# entire presentation template can be a single iw_perl
# tag if we wish!

for (my $i=0; $i<10; ++$i)
{
    iwpt_output("\nYou are on Diet Coke number: $i\n");
}

iwpt_output("<h3>Looping over multiple DCRs:</h3>\n");

```

```

#-----+
# Warning: Multiple DCRs are not supported in this version due
#         to limitations in the GUI and in iwgen/iwregen.
#
# To iterate through multiple DCRs, loop over the
#     top-level property, 'dcr'. Recall that 'dcr.xyz'
#     is implicitly the 'xyz' property of dcr[0]; 'dcr[0].xyz'
#     means the same thing as 'dcr.xyz'. Therefore, a bare
#     'dcr' is the list of all DCRs available.
#
#     NOTE: Follow the arrows to understand the relationship
#           between <iw_perl>'s API for DCRs and the
#           <iw_iterate> & <iw_value> tags.
#-----+

```

```

foreach my $index ( iwpt_dcr_list( 'dcr' ) )
{
    #         ^
    #         |

```

```

#      |                               |
#      +-----+-----+             +-----+
#                                       |         |
#                                       |         |
#                                       |         v
#      <iw_iterate var='index' list='dcr'>
#                                       |
#      <iw_value name='index.Headline' />
#                                       |
#      </iw_iterate>                    |
#                                       v
#
my $head = iwpt_dcr_value('index.Headline' ) ;

iwpt_output( "
                DCR headline:
                <font color='red'>
                <h2>    $head    </h2>
                </font>
                <p>
                "
                );

}
iwpt_output( "\n    <hr>    \n");

]]</iw_perl>

... whatever...
</iw_pt>

```

<iw_pt>

The <iw_pt> tag identifies a file as a presentation template and specifies the template's name. The <iw_pt> tag must enclose the entire presentation template except the XML declaration. It must be used exactly once in a presentation template containing tags derived from iw_xml.

For information on iw_pt flags, see “iwpt_compile.ipl” on page 372.

Example

The following code shows correct usage of the <iw_pt> tag:

```
<?xml version="1.0" standalone="yes"?>
<iw_pt>
...
</iw_pt>
```

The following code shows *incorrect* usage of the <iw_pt> tag:

```
<?xml version="1.0" standalone="yes"?>
<iw_repeat count=2> ... </iw_repeat>
<iw_pt>
...
</iw_pt>
```


<iw_repeat>

The `<iw_repeat>` tag allows you to repeat the content contained within the `<iw_repeat>` tag a given number of times. You may nest other `iw_xml` tags within the `<iw_repeat>` tag.

Attributes

This tag has one optional attribute, `count`.

`count='number'` Specifies the number of times to repeat the content.

Example

The following example repeats the string “Hello! ” six times in the generated HTML files:

```
<iw_repeat count="6">Hello! </iw_repeat>
```

The result of this code is:

```
Hello! Hello! Hello! Hello! Hello! Hello!
```

<iw_sql_iterate>

The <iw_sql_iterate> tag is used to iterate SQL result sets. See also <iw_sql_query> on page 118.

Attributes

The <iw_sql_iterate> tag has two mandatory attributes, `result_set` and `var`, and two optional attributes, `iteration` and `label`.

`var='variable'` Specifies the iterator variable that contains the current value within the `result_set` (Example 1).

`result_set='name'` Specifies the name of the SQL query result set being iterated through. Similar to the `list` attribute of <iw_iterate>.

`iteration='counter_name'` Overrides the name of the counter used by tags like <iw_if> to determine how many elements within the result set have been iterated through so far. The default counter name is `iw_sql_iteration`.

It is occasionally useful to override the default name if you wish to nest <iw_sql_iterate> tags and use conditional logic that depends on the values of outer and inner loop counters within the innermost looping structure. See <iw_iterate> (page 99) for details.

`label='label'` Allows <iw_next> (page 105) and <iw_last> (page 104) to jump out of deeply nested looping structures.

Note: When the `iw_value` tag accesses a column within an SQL row, it does so like this:

```
<iw_value name='$current_hat->{hat_size}' />
```

Here, the var for the `result_set` "hats" is named "current_hat" (current_hat corresponds to a database row).

Even if a column in the database is specified in upper case, it is always accessed via lower case within presentation templates (this eliminates database-to-database inconsistencies in the way case conversion is handled). For example, whether the `hat_size` column is present in the database as `Hat_Size` or `HAT_SIZE`, templates will refer to the database column as `hat_size`.

Example 1

```
<iw_sql_open data_source = "dbi:mysql:iw_demo"
              user_name  = "root">
  <iw_sql_query stmt      = 'select * from customer'
                result_set = 'all_customers' />
  <iw_sql_iterate var      = 'current_customer'
                 result_set = 'all_customers'
                 iteration  = 'nrow'>
    Here is a customer's first name in the database:
    <iw_value name='$current_customer->{first_name}' />
    It was on row: <iw_value name='$nrow' />
  </iw_sql_iterate>
</iw_sql_open>
```

Example 2

```
<iw_sql_iterate var='current_person' result_set='people'
                label='outerloop'>

  <iw_value name='$current_person->{hat_size}' />

  <iw_sql_iterate var='current_hat' result_set='hats'
                 iteration='nhats'>
    <iw_if expr='{iw_value name="nhat"/} == 13'>
      <iw_then>
        <!-- you can exit out of multiple levels! -->
```



```
        <iw_last label='outerloop' />
    </iw_then>
</iw_if>
    This hat is of size: <iw_value name='$current_hat
        ->{hat_size}' />
</iw_sql_iterate>
</iw_sql_iterate>
```

Note: On Windows NT/2000 systems, the `iw_sql` template tags require a database source name (DSN) to be configured using the ODBC control panel. The Default Network for the DSN should be set to TCP/IP rather than Named Pipes in order for Generate HTML or template preview to connect to the database. From the ODBC control panel, click the **System DSN** tab, select **LocalServer**, click **Configure**, click **Next**, and click **Client Configuration**. In the SQL Server Client Configuration Utility window, select the **Net Library** tab, and replace Named Pipes with TCP/IP.

<iw_sql_open>

The <iw_sql_open> tag is used to open a database connection.

Attributes

The <iw_sql_open> tag has one required attribute, `data_source`, and five optional attributes, `user_name`, `connection`, `password`, `aux_env`, and `attr`.

`data_source='string'` Specifies the DBI/DBD connection string used to specify the database and schema to which a connection is being made.

`user_name='name'` Specifies the account name used to make the database connection.

`password='password'` Specifies the authorization information for `user_name`.

`connection='name'` Specifies the name of the SQL connection. Defaults to `iw_sql_connection`. Explicitly naming the connection is useful if you wish to have multiple database connections open simultaneously (Example 2).

Note: When an <iw_sql_query> is using a database connection that is not the default (`iw_sql_connection`), it must provide the connection name explicitly.

`aux_env='variable'` Specifies the auxiliary environment variables needed to make the connection. Most database drivers do not need this, but Oracle requires `ORACLE_HOME` and `ORACLE_SID` to be set (Example 3).

`attr='attribute'` Specifies the auxiliary DBI/DBD flags for the database.
Default values:

```
" PrintError   => 0,
  RaiseError   => 0,
  AutoCommit   => 1,
  LongTruncOk  => 1,
  LongReadLen  => 3200,
  ChopBlanks   => 1,
"
```



Each key, value pair that you supply overrides the corresponding default. For example: `attr = "ChopBlanks => 0, LongReadLen => 64000"` results in:

```
" PrintError  => 0,
   RaiseError  => 0,
   AutoCommit  => 1,
   LongTruncOk => 1,
   LongReadLen => 6400,
   ChopBlanks  => 0,
"
```

Note: For information on Perl DBI/DBD, see *Programming the Perl DBI* by A. Descartes and T. Bunce, published by O'Reilly & Associates, ISBN: 1-56592-699-4.

Example 1

```
<iw_sql_open data_source      = "dbi:mysql:iw_demo">
  <iw_sql_query stmt          = 'select * from customer'
    result_set = 'all_customers' />
  <iw_sql_iterate var          = 'current_customer'
    result_set = 'all_customers'
    iteration  = 'nrow'>
```

Here is a customer's first name in the database:

```
<iw_value name='$current_customer->{first_name}' />
It was on row: <iw_value name='$nrow' />
</iw_sql_iterate>
</iw_sql_open>
```

Example 2

```
<iw_sql_open data_source = "dbi:mysql:iw_demo"
  user_name   = "root"
  password    = "rumplestiltskin"
  connection  = 'dbh'
  attr        = "RaiseError=> '0',
                AutoCommit=> '1'">
  <iw_sql_query stmt      = 'select * from customer'
    connection = 'dbh'
    result_set  = 'sth' />
```

```
        <iw_sql_iterate var='row' result_set='sth' >
            <iw_value name='$row->{property}'/>
        </iw_sql_iterate>
    </iw_sql_open>
```

Example 3

```
<iw_sql_open data_source = "dbi:Oracle:"
    user_name    = "system"
    password     = "manager"
    aux_env      = "ORACLE_HOME => '/opt/oracle/805',
                  ORACLE_SID  => 'newton'" >
```

```
    <iw_sql_iterate var          = 'current_customer'
                  result_set = 'all_customers'
                  iteration  = 'nrow'>
```

Here is a customer's first name in the database:

```
    <iw_value name='$current_customer->{first_name}'/>
```

It was on row: <iw_value name='\$nrow'/>

```
    </iw_sql_iterate>
</iw_sql_open>
```

<iw_sql_query>

The <iw_sql_query> tag is used for querying an SQL database.

Attributes

The <iw_sql_query> tag has two required attributes, `stmt` and `result_set`, and one optional attribute, `connection`.

`stmt = 'query'`

Specifies the SQL query being issued. The results of this query are stored in `result_set`. If `{iw_value name='...'/}` appears within the SQL statement, variable substitution takes place (see <iw_value> page 122).

Example 3 demonstrates both the use of `{iw_value name='...'/}` and placing the query `stmt` within the body of the <iw_sql_query> tag. This is useful when you have a long query and placing it in the attribute list is awkward. The statement may be supplied in an attribute list or the tag body.

`result_set = 'name'`

Specifies the name of the SQL query result set into which the database will return all rows matching the request.

`connection = 'name'`

Specifies the name of the SQL connection. Defaults to `iw_sql_connection`. Explicitly naming the connection is useful if you wish to have multiple database connections open simultaneously (Example 2).

Note: When an <iw_sql_query> is using a database connection that is not the default (`iw_sql_connection`), it must provide the connection name explicitly.

Example 1

```
<iw_sql_open data_source = "dbi:mysql:iw_demo"
            user_name   = "root">
  <iw_sql_query stmt      = 'select * from customer'
              result_set = 'all_customers' />
  <iw_sql_iterate var      = 'current_customer'
                result_set = 'all_customers'
```



```

        iteration = 'nrow'>

        Here is a customer's first name in the database:
        <iw_value name='$current_customer->{first_name}' />
        It was on row: <iw_value name='$nrow' />
    </iw_sql_iterate>
</iw_sql_open>

```

Example 2

```

<iw_sql_open connection = 'dbh'
              data_source = 'dbi:mysql:iw_demo'
              user_name   = 'root'
              auth        = 'rumplestiltskin'
              attr        = 'RaiseError , 1,
                          AutoCommit , 1  '>

    <iw_sql_query stmt      = 'select * from customer'
                 connection = 'dbh'
                 result_set = 'rs' />

    <iw_sql_iterate var='row' result_set='rs'>
        <iw_value name='$row->{property}' />
    </iw_sql_iterate>

</iw_sql_open>

```

Example 3

```

<iw_sql_open data_source = "dbi:mysql:iw_demo"
              user_name   = "root">

    <iw_sql_query result_set = 'folks_from_a_company'>
        select * from customer
        where company
        like "{iw_value name='$iw_arg{company_name}' /}"
    </iw_sql_query>

    ... whatever...

</iw_sql_open>

```

<iw_system>

The `<iw_system>` tag inserts the output of an external program into the generated page.

Attributes

The `<iw_system>` tag has one mandatory attribute, `command`, and one optional attribute, `ienc`.

<code>command='command'</code>	Specifies the command line to be executed. The output of this command is inserted into the generated HTML page.
<code>ienc='encoding'</code>	Specifies encoding. When the result of the command line's execution is non-UTF-8 data, its encoding must be specified via the <code>ienc</code> attribute.

Example

The following example shows proper usage of the `command` attribute.

```
<iw_system command='ls -l' />
```

<iw_then>

The <iw_then> tag is used for conditional inclusion of contents. It is used with the <iw_if> tag (page 89) and provides contents to be included if the <iw_if> tag is true. The <iw_else> tag (page 88) is also used and provides the contents if the <iw_if> tag is false.

Examples

The following examples take this form:

```
<iw_if expr='  some logical condition ' >

    <!-- optional then clause (only included when iw_if is true) -->
    <iw_then>...</iw_then>

    <!-- optional else clause (only included when iw_if is false) -->
    <iw_else>...</iw_else>

</iw_if>
```

Example 1

```
<iw_if expr=' {iw_value name="$iw_arg{moo}"/} eq "cow" ' >
  <iw_then>
    do this if the condition is true
  </iw_then>
  <iw_else>
    do this if the condition is false
  </iw_else>
</iw_if>
```

Example 2

```
<iw_if expr=' ( {iw_value name="dcr.xyz"/} > 42 )
              ( {iw_value name="$iw_arg{pdq}"/} < 99 )
              '>
  <iw_then>
    do this if the condition is true
  </iw_then>
</iw_if>
```

<iw_value>

The <iw_value> tag allows you to insert a Perl value or a value from a data content record where this tag appears.

Attributes

The <iw_value> tag has one required attribute, *name*, and one optional attribute, *ienc*.

name='variable_name' Specifies the name of the value being put into the generated document. If *variable_name* begins with a letter or underscore, the variable being referred to resides in a data content record. If *variable_name* is undefined, a 0-length string is inserted.

name	corresponds to
dcr.x.y	y component of top-level DCR/XML node x
hen.egg	the egg XML node within the hen node
cow@moo	moo attribute of DCR/XML node cow
z	the Perl variable \$z
\$z	the Perl variable \$z
(. . . .)	the result of the Perl expression

ienc='encoding' Specifies current encoding (input encoding) when the variable specified by the *name* attribute is not UTF-8 normalized. Mandatory when *name* is not UTF-8 normalized. *ienc* is useful when collecting data from sources other than data content records (for example, a database whose content is encoded in BIG5).

Example 1

If the value of the 0th headline in a data content record is “Dewey Wins!”, the following:

```
Flash: <iw_value name='dcr.Headline' /> Read all about it!
```

compiles into:

```
Flash: Dewey Wins! Read all about it!
```

Example 2

Every component within a data content record may have a list of subcomponents. Therefore, all references to subcomponents within a data content record, denoted by “.” (a period) in the `name` attribute, are implicitly subscripted by `[0]` unless an explicit subscript is provided.

```
<iw_value name='dcr.hen.egg' />
```

is the same as:

```
<iw_value name='dcr[0].hen[0].egg[0]'>
```

This means that the `<iw_value>` tag supports immediate access into any part of a data content record.

Example 3

If the `variable_name` begins with a `$`, the data being referred to is a Perl scalar.

Therefore, if the value of the Perl variable `$person` is `Jon`, the following line:

```
Hi there <iw_value name='$person' />, how are you?
```

compiles into:

```
Hi there Jon, how are you?
```



If the `variable_name` is surrounded by parentheses, it is treated like a Perl expression, which makes it easy to call functions and evaluate formulas inline. Therefore:

```
A day has <iw_value name='( 60*60*24 )' /> seconds
```

compiles into:

```
A day has 86400 seconds!
```

Example 4

In cases where `name` consists only of a letter or underscore followed by letters, digits, and/or underscores, preceding it by a single `$` does not matter. Therefore, when this code is used:

```
<iw_perl><![CDATA[ $person = 'Jon'; ]]></iw_perl>
```

the following two lines result in equivalent `name` values:

```
<iw_value name='$person' />
<iw_value name='person' />
```

However, the following line is illegal:

```
<iw_value name='$x.y' />
```

because the `$` implies that `x.y` is a Perl variable—`x.y` is not a legal Perl variable name!

Example 5

By default, the value of an XML node is the value of the 0th character data component of the node. You may explicitly change this default.

Given XML like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<hen>
  <egg>this           <yolk>is an       </yolk>advanced</egg>
  <egg>example       <yolk>involving  </yolk>explicit</egg>
  <egg>character data <yolk>accessor   </yolk>strings!</egg>
</hen>
```

the following line:

```
<iw_value name='dcr.hen.egg[1]'/>
```

emits example , as does this:

```
<iw_value name='dcr.hen.egg[1].character data[0]'/>
```

However this line:

```
<iw_value name='dcr.hen.egg.character data[1]'/>
```

emits advanced and this line:

```
<iw_value name='dcr.hen.egg[2].character data[1]'/>
```

emits strings!

Example 6

The `<iw_value>` tag is unusual because it has two distinct forms:

```
<iw_value name='...' />
```

and when in a CDATA section, or in various attributes of certain tags,

```
{iw_value name='...'/}
```

The second form exists because XML does not allow tags inside of attribute lists; however, sometimes this is what you need to make coding simpler. Therefore, the following line is permissible:

```
<iw_if expr=' {iw_value name="$nrow"} == 3' > ...
```

Example 7

Both the `<iw_value name='...'/>` tag form and the `{iw_value name='...'/}` tag form work within CDATA sections. This makes it easy to do things like emit values within a CDATA section containing HTML, as shown in the following code:

```
<some_arbitrary_tag><![CDATA[
```

```
    It's ok to use the value tag inside a CDATA section
```

```
    <iw_value name='dcr.x.y'/>    this works!
```

```
    {iw_value name='dcr.x.y'/}    this does too (it means the same thing)!
```

```
    <p>
```

```
    That was fun!
```

```
]]>
```

```
</some_arbitrary_tag>
```

Synopsis

```
<iw_pt>
```

```
    Explicit dcr value: <iw_value name='dcr.stuff[0]'/>
```

```
    Explicit dcr value: <iw_value name='dcr.stuff[1]'/>
```

```
    Implicitly specified dcr value: <iw_value name='dcr.stuff' />  
(note: the default index is always 0).
```

Access a DCR value through an iterator variable ('current_dcr'):

```
    <iw_iterate list='dcr' var='current_dcr'>  
        <iw_value name='current_dcr.Headline' />  
    </iw_iterate>
```

Access Perl data from XML:

```
<iw_perl><![CDATA[  
  
    $jcox_haiku = "Coffee overload          \n" .  
                "Roller-coaster dosage ride \n" .  
                "Fast, slow, giddy , low.  \n" ;  
  
    $data_encoded_in_BIG5 = '... big5 content...';  
  
    ]]></iw_perl>  
  
<iw_value name='$jcox_haiku' />  
<iw_value name='$data_encoded_in_BIG5' ienc='BIG5' />
```

```
</iw_pt>
```


Writing Your Own Tags

If the provided `iw_xml` tags do not satisfy your needs, it is possible to write your own custom XML tags. However, this is an advanced exercise.

For assistance in debugging tags and presentation templates, use the `-ocode` flag on the `iwpt_compile` command. Additional debugging help is available with the command `/iw-home/iw-perl/bin/iwperl -w`. You can invoke a graphical interface with the command `/iw-home/iw-perl/bin/iwperl -d:ptkdb`.

Mapping Users, Templates, and Content Records

This chapter describes how the `templating.cfg` file maps the interaction between content contributors, data capture templates, presentation templates, and data content records. Sections in this chapter discuss the following:

- An overview of `templating.cfg`.
- Pointers to sample `templating.cfg` files that are included with this release of TeamXpress Templating
- An example of a `templating.cfg` file.
- The `templating.cfg` DTD.

templating.cfg Overview

The `templating.cfg` file is an XML file that resides outside of the TeamXpress file system in `iw-home/local/config`. Each TeamXpress Templating installation must have exactly one such file. By configuring `templating.cfg`, you can control:

- Which data categories and types TeamXpress Templating is aware of.
- Which presentation templates can generate HTML files on which branches and/or directories.
- Which presentation templates can be used with a specific data type.
- Which users or roles are allowed to create or edit data content records for a specific data type.
- The location of the presentation template used for previewing generated HTML files.

The following sections describe how to perform these configurations.

Example templating.cfg File

TeamXpress Templating ships with the following sample `templating.cfg` file:

```
iw-home/examples/Templating/config/templating.cfg.example
```

This is the `templating.cfg` file that configures the example templating environment described in Chapter 2, “Initial Configuration.” It configures TeamXpress Templating to recognize and use the following data categories/types:

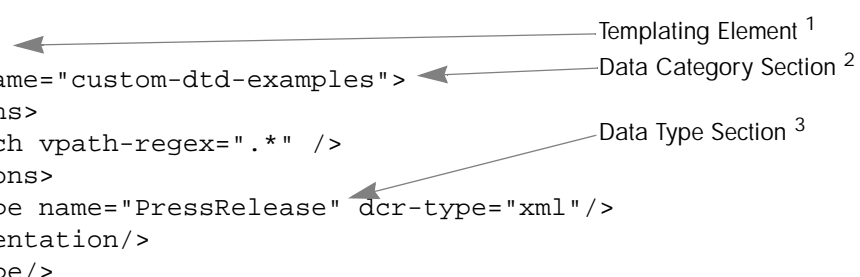
```
custom-dtd-example/PressRelease
intranet/deptinfo
intranet/weather
internet/careers
internet/auction
internet/pr
internet/book
internet/medical
internet/periodic
```

The following section shows a subset of this file with sections for the `PressRelease`, `deptinfo`, `weather`, `careers`, and `auction` data types.

See the diagram key on page 132 for details about each item referenced in the following diagram.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE templating SYSTEM "templating4.5.dtd">

<templating>
  <category name="custom-dtd-examples">
    <locations>
      <branch vpath-regex=".*" />
    </locations>
    <data-type name="PressRelease" dcr-type="xml"/>
      <presentation/>
    </data-type/>
```



```

</category>
<category name="intranet">
  <locations>
    <branch vpath-regex=".*" />
  </locations>
  <data-type name="deptInfo" dcr-type="iwov">
    <presentation>
      <template name="deptInfo.tpl" extension="html">
        <locations>
          <branch vpath-regex=".*" preview-dir="/">
            <directory dir-regex=".*" />
          </branch>
        </locations>
      </template>
    </presentation>
  </data-type>
  <data-type name="weather" dcr-type="iwov">
    <presentation>
      <template name="weather.tpl" extension="html">
        <locations>
          <branch vpath-regex=".*" preview-dir="/">
            <directory dir-regex=".*" />
          </branch>
        </locations>
      </template>
    </presentation>
  </data-type>
</category>
<category name="internet">
  <locations>
    <branch vpath-regex=".*" />
  </locations>
  <data-type name="careers" dcr-type="iwov">
    <presentation>
      <template name="jobDesc.tpl" extension="html">
        <locations>
          <branch vpath-regex=".*" preview-dir="/">
            <directory dir-regex=".*" />
          </branch>
        </locations>
      </template>
    </presentation>
  </data-type>
</category>

```

Data Category Section ²

Data Type Section ³

Presentation Template Section ⁴

Template to Data Type Mapping⁵

Template to Generated File Mapping ⁶

Generated HTML File Location ⁷



```
</presentation>
</data-type>
<data-type name="auction" dcr-type="iwov">
  <presentation>
    <template name="auction.tpl" extension="html">
      <locations>
        <branch vpath-regex=".*" preview-dir="/">
          <directory dir-regex=".*" />
        </branch>
      </locations>
    </template>
  </presentation>
</data-type>
</category>
</templating>
```

Diagram Key

- 1. Templating Element:** The `<templating>` element marks the beginning of the `templating.cfg` file's configuration information and identifies the file as a `templating.cfg` file.
- 2. Data Category Section:** The `<category>` element contains information specific to a data category (`intranet` in this example) and makes the data category available for use by TeamXpress Templating. The `<category>` element contains one or more `<data-type>` elements. A data category must have its own `<category>` element in `templating.cfg` in order for TeamXpress Templating to recognize and use the data category. Even if a data category is located correctly in the directory structure described on page 21, it will not be recognized by TeamXpress Templating unless it is named in a `<category>` element as shown here. The `<category>` element's name attribute is required. You can use the `<locations>` element within a `<category>` element to show the branches in which that category will be available. This example shows that `intranet` category will be available in all branches.
- 3. Data Type Section:** The `<data-type>` element contains information specific to a data type (`careers` in this example) and makes the data type available for use by TeamXpress Templating. A data type must have its own `<data-type>` element in `templating.cfg` in order for TeamXpress Templating to recognize and use the data type. Even if a data type is located correctly in the directory structure described on page 21, it will not be recognized by TeamXpress Templating

unless it is named in a `<data-type>` element as shown here. The attributes for `<data-type>` are `name` and `dcr-type`. The `<data-type>` element's `name` attribute is required. The `dcr-type` specifies what kind of DCR to write out. The values are `xml` and `iwov`; the default is `iwov`. If the value of `dcr-type` is `xml`:

- The data capture template for that data type needs to have been generated using `iwtdtd2sym` and `iwsym2dct`.
- The data content records for that data type will be XML documents written according to the DTD that the data capture template was derived from.
- You must use the Java-based interface. See "Modifying the TeamXpress iw.cfg File" on page 32 for information on enabling it.

The `<data-type>` element can contain the following subelements:

- `<locations>`: Shows the branches in which that data type will be available.
- `<presentation>`: See Item 4 below.
- `<allowed>`: Lets you set an ACL to specify which users can or cannot use a specific data type. If `<allowed>` is not set, any user can use the data type (see page 53 for additional examples). The `<allowed>` element can have any of the following subelements:

<code><cred></code> :	Lets you name a user or role in the ACL (e.g., <code>user="joe"</code> or <code>role="master"</code>).
<code><and></code> :	Logical <code>and</code> statement for grouping ACL credentials.
<code><or></code> :	Logical <code>or</code> statement for grouping ACL credentials.
<code><not></code> :	Logical <code>not</code> statement for negating ACL credentials. For example, the following allows all users except <code>joe</code> to use the current instance:

```
<allowed>
  <not>
    <cred user="joe">
    </cred>
  </not>
</allowed>
```

- 4. Presentation Template Section:** The `<presentation>` element marks the beginning of the section that contains subelements for presentation template mapping. See Items 5, 6, and 7 below.



5. **Template to Data Type Mapping:** The `<template>` element marks the beginning of the section that maps a presentation template to a data type. It specifies which presentation templates are available for use with the data type named in the `<data-type>` element. In the example shown here, the `deptInfo.tpl` template can be used to display data content records for the `deptinfo` data type. The `<template>` element can contain the following attributes:
 - `extension`: Specifies the extension that will be used on any files this template generates. This attribute is required.
 - `fullpage`: Specifies that the generated HTML file is a full atomic HTML page. This attribute is optional.
 - `name`: Specifies the presentation template's file name in the `workarea_name/templatedata/data_category/data_type/presentation` directory. This attribute is required.
6. **Template to Generated File Mapping:** The `<branch>` element uses Perl regex (Perlre) syntax to specify on which branches a presentation template can generate a file. The `<branch>` element can have the following attributes:
 - `vpath-regex`: Specifies on which branch(es) files can be generated via this presentation template. The example shown here ("`. *`") specifies that all branches can have files generated via the `deptInfo.tpl` presentation template.
 - `preview-dir`: Specifies what directory (in an area of a branch) generated files will be previewed in when you preview a data content record (via the TeamXpress GUI's **Save and Preview** button).
7. **Generated HTML File Locations:** The `<directory>` element uses regex syntax to specify where generated HTML files based on this presentation template will reside. This example specifies that generated HTML files based on `jobDesc.tpl` will reside in the current directory (`. *`).

The `<directory dir-regex="..." />` regular expression matches a directory relative to the user's workarea. Because the string that is matched against the regex does not begin with a slash, it is possible for the string to be empty (i.e., when the directory in question is the top of the workarea, then an empty string will be matched against the regex).

When **Generate/Preview** is selected when you are creating or editing a data capture form, only Presentation Templates with a `dir-regex` entry that matches the workarea root, as identified by the `<directory>` element, appear on the Presentation Template selection list.

Setting Previewing Path Variables

The following example describes what happens when a user previews a generated HTML file in TeamXpress Templating.

If the file is specified with an absolute path (e.g., `href=/main/dev/images/pixel.gif`), the browser searches the absolute path.

The way to configure TeamXpress Templating so that the correct directory is searched is to set `preview-dir` in the `templating.cfg` file to point to the directory containing the file. For example, set the `preview-dir` variable to `/images` if `pixel.gif` resides in `/images`. Then `pixel.gif` will be found and displayed during the preview.

To summarize the preview results:

- If the line `href=pixel.gif` appears in the presentation template and the directory containing `pixel.gif` is named with the `preview-dir` variable in `templating.cfg`, `pixel.gif` will be included in the preview.
- If the line `href=absolute_path_name/pixel.gif`, the file `pixel.gif` will be included in the preview.

The `preview-dir` variable (in the `templating.cfg` file) associated with each presentation template defines the directory where the preview file will virtually exist during preview time. When a preview occurs, a temporary file is created in the `templatedata/iw_preview` directory. However, when a browser is opened and directed to the preview file, the URL that the browser points to is the URL for the preview file in the directory defined in `preview-dir`. During the preview, a proxy remap occurs, remapping the directory specified in the `preview-dir` variable to the `templatedata/iw_preview` directory. In this way, a preview file can have a virtual location other than its true location. These temporary files are deleted by the previewing system, based on the file's modification time. The default is files that have not been modified in the last 60 minutes are deleted at preview time. If you do not do another preview, the files will not be deleted. You can change the time in the `iw.cfg` file (see "Saving Preview Files" on page 32).

templating.cfg DTD

```
<!ELEMENT templating (category*) >

<!ELEMENT category (locations?,data-type*) >
  <!ATTLIST category
    name          CDATA          #REQUIRED
  >
<!ELEMENT data-type (locations?,allowed?,presentation+) >
  <!ATTLIST data-type
    name          CDATA          #REQUIRED
    dcr-type      (iwov|xml)    "iwov"
  >

<!ELEMENT presentation (template*) >

<!ELEMENT template (locations) >
  <!ATTLIST template
    name          CDATA          #REQUIRED
    fullpage      t|f)          "f"
    extension     CDATA          #REQUIRED
  >

<!ELEMENT locations (branch+) >

<!ELEMENT branch (directory*) >
  <!ATTLIST branch
    vpath-regex   CDATA          #REQUIRED
    preview-dir   CDATA          #IMPLIED
  >

<!-- 'branch' elements should only contain 'directory' elements
when they are within a 'template' element.
The 'preview-dir' attribute is required when the 'branch' element
is within a 'template' element. -->

<!ELEMENT directory EMPTY >
  <!ATTLIST directory
    dir-regex     CDATA          #REQUIRED
  >
```

```
<!-- This is the same stuff as datacapture4.5.dtd: -->  
  
<!ELEMENT allowed (cred|and|or|not) >  
  
<!ELEMENT cred EMPTY >  
    <!ATTLIST cred  
        role          CDATA          #IMPLIED  
        user          CDATA          #IMPLIED  
    >  
  
<!ELEMENT and (cred|and|or|not)+ >  
  
<!ELEMENT or (cred|and|or|not)+ >  
  
<!ELEMENT not (cred|and|or|not) >
```


Integrating Templating, DataDeploy, and Workflow

This chapter describes how to integrate TeamXpress Templating with DataDeploy and TeamXpress workflow. Integrating these components lets a content contributor access a data capture template, create a data content record, and deploy the data content record's extended attributes to a database via a TeamXpress workflow job. All of these activities take place as a single, integrated sequence of steps initiated and executed from the TeamXpress GUI. The entire DataDeploy process runs as a TeamXpress workflow job, so the content contributor does not need to start DataDeploy manually, or even be aware that DataDeploy is running.

Note: The configuration steps described in this chapter assume that TeamXpress Templating is already installed and configured as described in Chapter 2, "Initial Configuration." DataDeploy must also be installed on your system.

Refer to the *TeamXpress User's Guide* for information on using templating. Refer to the *TeamXpress Administration Guide* for information on setting up TeamXpress.

Integration Overview

The following steps show the process flow for creating, saving, submitting, and deploying a data content record when TeamXpress Templating and DataDeploy are integrated.

1. In the TeamXpress GUI, a content contributor selects **File > New Data Record**, chooses a data type, and enters data in the resulting data capture form.
2. The content contributor selects **File > Save** in the data capture form.

3. In the TeamXpress GUI, the content contributor navigates to the data type's `data` directory, selects a data content record, and clicks **Submit**. Templating can be configured to automatically initiate a workflow process upon **Save** as a convenience to the end user. This can be done in `available_templates.ipl` (see “Editing `available_templates.ipl` to Initiate Workflows” on page 31).
4. DataDeploy is automatically signaled to perform the following functions:
 - Determine which data types are affected by the data content record change.
 - Read in all necessary database mapping information from DataDeploy configuration files.
 - Populate the database with some or all of elements of the data content record, based on the mapping file.
 - Write a log of all DataDeploy activity to the `dd-home/log` file.

Integration Steps

The following sections describe the configuration steps you must perform on TeamXpress Templating, TeamXpress workflow, and DataDeploy to integrate them for your specific templating environment.

Integration Steps: TeamXpress Templating

Installing and setting up TeamXpress Templating as described in Chapter 2, “Initial Configuration,” prepares TeamXpress Templating for integration with DataDeploy and TeamXpress workflow. You do not need to perform any additional tasks on TeamXpress Templating to enable integration.

Integration Steps: DataDeploy

A DataDeploy configuration file must be created for each type of data content record that will be deployed. DataDeploy generates these configuration files automatically. However, the information is provided here for your information. For example, to use DataDeploy to deploy a data content record that is based on the data capture template `/templatedata/beverages/tea/datacapture.cfg`, a DataDeploy configuration file must be created specifically for the data type `tea`. Likewise, to deploy a data content record based on `/templatedata/beverages/coffee/datacapture.cfg`, a DataDeploy configuration file must be created specifically for the data type `coffee`.

By default, DataDeploy configuration files for TeamXpress Templating use the following location and naming conventions:

```
workarea_name / templatedata / data-category / data-type / data-type_dd.cfg
```

For example:

```
/workarea_name / templatedata / beverages / tea / tea_dd.cfg
```

Or, in the case of the Press Release example shown in “Data Capture Example 1” on page 39:

```
/workarea_name / templatedata / internet / pr / pr_dd.cfg
```

Refer to Appendix C, “DataDeploy Database Auto-Synchronization” for information on creating the DataDeploy configuration files and the database tables.

Integration Steps: TeamXpress Workflow

This release of TeamXpress Templating supports a preconfigured templating-specific workflow template, `author_submit_dcr.wft`. This file is distributed with TeamXpress Templating in `iw-home/examples/Templating/workflow`. It configures the **Author DCR Submit** workflow job displayed in the New job window when TeamXpress Templating starts a workflow job. Check the `available_templates.ipl` to verify that the workflow is set up and to add additional workflows. See Chapter 5 of the *TeamXpress Administration Guide* for an example of the TeamXpress GUI’s New job window.

Section 2: DataDeploy Administration

-
- Overview and Installation
 - Deployment Concepts
 - Configuration File Details and Examples
 - Invoking DataDeploy
 - Synchronizing OpenDeploy and Data Deploy

Chapter 7

Overview and Installation

Overview

DataDeploy lets you transfer extended attribute data between TeamXpress, an external SQL database, and an XML file. The following table shows which source/destination scenarios are supported:

		Destination		
		TeamXpress	XML	Database
Source	TeamXpress	Not Supported	Supported	Supported
	XML	Not Supported	Supported	Supported
	Database	Not Supported	Supported	Supported

Supported Data Sources and Destinations

There are several ways that you can configure and execute deployment:

- For all of the supported scenarios shown above, you can manually edit a DataDeploy configuration file and then execute DataDeploy from the command line. See “Invoking DataDeploy” on page 207 and “Configuration File Details and Examples” on page 171 for more information. Even though manual configuration and deployment is the least common way to use DataDeploy, the information in these sections provides a good background for understanding the more commonly used automated deployment described in the following bullets.
- Alternatively, after a DataDeploy configuration file exists for any supported scenario, you can automate DataDeploy execution by initiating it from an `iwat` trigger script or as a TeamXpress workflow task. For example, you can configure DataDeploy and TeamXpress so that extended

attribute data is transferred from TeamXpress to a database whenever files are submitted to a staging area from a workarea. See “Deployment Scenarios” on page 158 for more information about that specific situation.

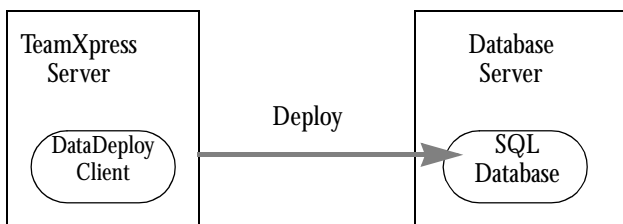
- For the TeamXpress-to-database scenario, you can use DataDeploy’s Database Auto-Synchronization (DAS) module to automate the entire deployment process for TeamXpress templating users. In this situation, any extended attribute changes resulting from an end user modifying a data content record via the TeamXpress templating GUI are automatically deployed to a database. See Appendix C, “DataDeploy Database Auto-Synchronization” for details about DAS.

Note: You can configure DataDeploy to treat deployed extended attributes as either data or stored procedures. See Item 10, “DataBase Section” in “Sample File Notes” starting on page 181 for more information.

The following sections describe overall DataDeploy concepts, and how you can install, configure, and invoke DataDeploy.

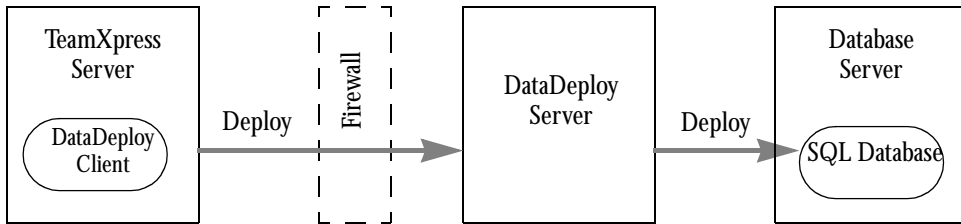
Client/Server Setup Options

When deploying to a database, you can set up DataDeploy to operate in either a *two-tier* or *three-tier* architecture. Two-tier architecture incorporates two systems: the TeamXpress server host machine that executes the DataDeploy client and an application server containing the SQL database. The application server can be any server on the network (such as the production web server, although this is not a system requirement). Two-tier systems are typically used at sites that do not require firewall protection between the TeamXpress server and the application or production server.



Two-Tier Architecture

Three-tier architecture incorporates a third system acting as a DataDeploy server. Three-tier systems are typically used at sites requiring firewall protection between the TeamXpress server and the application or production web server.



Three-Tier Architecture

Running the DataDeploy Daemon as a Service

You can optionally configure the `Interwoven DataDeploy` service to start the DataDeploy daemon for 3-tier operation or Database Auto-Synchronization (DAS) operation. When set up for 3-tier operation, the DataDeploy daemon takes its input from the `iwdd.ipl` command issued from the command line. When set up for DAS operation, the DataDeploy daemon takes its input from the `iwsyncdb.ipl` script that runs as part of DAS startup. See “Editing `iwsyncdb.cfg`” on page 386 for information about specifying DAS or 3-tier operation. See “Invoking DataDeploy” on page 207 for details about executing `iwdd.ipl` from the command line. See “Running `iwsyncdb.ipl`” on page 387 for details about `iwsyncdb.ipl`.

The `Interwoven DataDeploy` service automatically starts the DataDeploy daemon for DAS operation if the `iwsyncdb.cfg` file exists in `dd-home/conf`. If `iwsyncdb.cfg` does not exist, the `Interwoven DataDeploy` service starts the DataDeploy daemon for 3-tier operation.

Installing DataDeploy

Supported Platforms

DataDeploy supports the following database/platform combinations. Any DataDeploy server shown in the first column can run with any database server shown in the second column. Databases shown in the third column are specific to the servers shown for that row in the second column. For example, a system can have a Windows NT/2000 DataDeploy server and a Solaris database server. However, Microsoft SQL Server databases are not supported on Solaris database servers.

DataDeploy Server	Database Server Platform	Database (running on database server)	CPU
Windows NT (x86) 4.0 (Service Packs 3, 4, 5, 6a) Windows 2000 Server Solaris 2.5.1, 2.6, or 7 (32-bit and 64-bit)	Windows NT (x86) 4.0 (Service Packs 3, 4, 5, 6a)	IBM DB2 (UDB) 6.1; Microsoft SQL Server 6.5, 7.0; Oracle 8i *; Sybase SQL Anywhere 5.5; Sybase ASE 11.5; Informix 7.3 **	300 MHz Pentium II
	Windows 2000	IBM DB2 (UDB) 6.1; Microsoft SQL Server 6.5, 7.0; Oracle 8i *; Sybase SQL Anywhere 5.5; Sybase ASE 11.5; Informix 7.3 **	300 MHz Pentium II
	Solaris 2.5.1, 2.6, or 7 (32-bit and 64-bit)	Oracle 8i *; Sybase SQL Anywhere 5.5; Sybase ASE 11.5; Informix 7.3 **	Ultra5 SPARC

* If you use standard SQL datatypes, no additional Oracle products need to be installed. If you use Oracle extension datatypes you must also install the OCI client library on the system from which the `iwdd.ip1` command is executed.

** Only locally on the same system running DataDeploy.

Additional Drivers for Microsoft SQL Anywhere

To use DataDeploy with Microsoft SQL Anywhere, you must ensure that ODBC driver version 3.70.06.23 or later is installed.

Solaris Systems

Perform the following steps to install DataDeploy on a Solaris system:

1. Go to one of the following directories:
 - If TeamXpress is installed but OpenDeploy is not, go to `iw-home`.
 - If both TeamXpress and OpenDeploy are installed, go to `od-home`.
 - If OpenDeploy is installed but TeamXpress is not, go to `od-home`.
 - If neither TeamXpress nor OpenDeploy is installed, go to an installation directory of your choice.

2. Unzip and untar the DataDeploy tar file `dd.tar.gz`:

```
gunzip < dd.tar.gz | tar -xvpf -
```

An `opendeploy` directory and its associated subdirectories are created if they do not already exist in the directory from Step 1.

3. Go to the `opendeploy/install` directory and execute the `iwinstalldd` installation script.

Windows NT/2000 Systems

Perform the following steps to install DataDeploy on a Windows NT/2000 system:

1. Download the DataDeploy bundle from its distribution media. If the file is zipped, unzip it.
2. Double click the DataDeploy bundle icon.
3. If OpenDeploy is already installed on your system:
 - The uninstall (administrative) files should reside in a `dd-home` location of your choice. This location must be different than `od-home`.
 - All other DataDeploy system files are installed automatically in `od-home`.

If OpenDeploy is not installed on your system:



- The uninstall (administrative) files should reside in a *dd-home* location of your choice. This location must be different than *iw-home*.
- All other DataDeploy system files are installed automatically in *iw-home*.

After DataDeploy is installed, you might need to resynchronize the tracker table. Refer to the preceding section about installing on Solaris systems to determine whether this procedure is necessary.

Chapter 8

Deployment Concepts

This chapter describes the following general deployment concepts and components:

- How different methods of invoking DataDeploy affect the configuration activities you must perform.
- The roles and components of DataDeploy configuration files.
- How DataDeploy stores and processes data during a deployment.
- What happens during a TeamXpress-to-database deployment.

It is highly recommended that you understand the concepts in this chapter prior to configuring DataDeploy.

Ways to Invoke Deployment

There are several ways in which to invoke DataDeploy:

- From the command line.
- From an `iwat` trigger script.
- As a TeamXpress workflow task that is not associated with TeamXpress Templating.
- From the TeamXpress Templating GUI.

All of these methods require the existence of one or more DataDeploy configuration files. The first three methods require that you manually create these configuration files. The last method creates all

the necessary DataDeploy configuration files automatically after you have performed the necessary system setup. The following table shows a summary of each invocation method and its related tasks:

Invocation Method	Setup and Invocation Tasks	For More Information See...
Command Line	<ul style="list-style-type: none"> Manually create a DataDeploy configuration file. Execute iwdd.ipl from the command line. Deployment occurs when the command is executed. 	“Configuration File Details and Examples” on page 171; “iwdd.ipl Command” on page 207.
iwat Trigger Script	<ul style="list-style-type: none"> Manually create a DataDeploy configuration file. Create an iwat trigger script containing an iwdd.ipl command that references the DataDeploy configuration file. Deployment occurs when the iwat script is triggered. 	“Configuration File Details and Examples” on page 171; “iwdd.ipl Command” on page 207.
Workflow Task	<ul style="list-style-type: none"> Manually create a DataDeploy configuration file. Create a workflow external task containing an iwdd.ipl command that references the DataDeploy configuration file. Deployment occurs when the external task is executed. 	“Configuration File Details and Examples” on page 171; “iwdd.ipl Command” on page 207; “Configuring TeamXpress Workflow” in the <i>TeamXpress Administration Guide</i> .
TeamXpress Templating GUI	<ul style="list-style-type: none"> Install TeamXpress Templating. Configure Database Auto-Synchronization (DAS). Deployment occurs automatically whenever an end user modifies a data content record (DCR) via the TeamXpress Templating GUI. 	“DataDeploy Database Auto-Synchronization” on page 383

Configuration Files

DataDeploy configuration files let you specify the following:

- What, where, and how data is deployed.
- Whether DataDeploy will run as a client or server.

A TeamXpress/DataDeploy installation can contain any number of configuration files. The most common scenario is for a system to contain multiple configuration files, one for each specific type of deployment.

For the “TeamXpress Templating GUI” scenario shown in the preceding table, a DataDeploy configuration file is automatically created for each data type in the TeamXpress Templating directory structure. The correct configuration file is then referenced automatically whenever an end user changes a DCR for a given data type.

For the other scenarios shown in the preceding table, you must create each configuration file manually, and then name the correct file via a command line option for the `iwdd.ip1` command.

File Components

All DataDeploy configuration files have the following characteristics:

- Can have any name.
- Are in XML format.
- Reside by default in the `dd-home/conf` directory.

A configuration file is structured as a hierarchy of sections, each letting you control a different deployment parameter. A file can have any number of sections. Parameters that you can set are:

- Filters that include and exclude possible data sources.
- Substitution rules to replace text and data values automatically during deployment.
- Client-specific parameters and activities.
- Type of deployment (TeamXpress-to-database, XML-to-database, and so on).
- Source of extended attribute data (TeamXpress, a database, or an XML file).
- Destination of extended attribute data (a database or an XML file).
- Details about source and destination data (specific fields to select, type of table to update or create, and so on).
- SQL commands that execute automatically during deployment.
- Server-specific parameters (for 3-tier systems).

See the sample configuration file sections starting on page 175 for details about configuration file structure and syntax.

See “Invoking DataDeploy” on page 207 for more information about configuring DataDeploy as a client or server. See “Configuration File Details and Examples” on page 171 for more information about controlling all other DataDeploy parameters.

Data Organization Within DataDeploy

When extended attribute data is deployed, it is first extracted from its specified source and represented internally in DataDeploy as *tuples*. Tuples can then be deployed into a specified destination using selection and formatting rules defined in the DataDeploy configuration file(s). You can set tuple format to *narrow* or *wide*. The following section describes tuple format in general, and the differences between narrow and wide tuples.

Tuple Format

All TeamXpress tuples contain the following extended attribute data:

- Exactly one *path* element, which is an area-relative path name of the file associated with the tuple’s key-value pair(s).
- One or more *key/value pairs*. The key is the name (also known as the *class*) of the extended attribute. For example, `News-Section` is the key of the extended attribute `News-Section:Sports`. The value is the data value for tuple’s key. For example, `Sports`.
- Exactly one *state* element, which describes the status of the tuple. Possible values are `Original`, `New`, `Modified`, and `NotPresent`. See “Data Destinations” on page 159 for details about each state value.

The following sections describe how elements are arranged within narrow and wide tuples.

Narrow Tuples

Narrow tuples contain exactly one path, key, value, and state element. For example, the following figures show DataDeploy’s internal representation of two narrow tuples. Tuple 1 is for the `News-Section:Sports` extended attribute from the file `docroot/news/front.html`. Tuple 2 is for

the `Locale:SF` extended attribute from the same file. Note that because a narrow tuple can contain just one `key/value` pair, DataDeploy must create multiple tuples (two in this case) if a file's extended attributes consist of more than one `key/value` pair.

Tuple 1	Tuple 2
<code>path = docroot/news/front.html</code>	<code>path = docroot/news/front.html</code>
<code>key = News-Section</code>	<code>key = Locale</code>
<code>value = Sports</code>	<code>value = SF</code>

Narrow Tuples

Wide Tuples

Wide tuples contain exactly one path element and one state element, and any number of `key/value` pairs. Thus, a file's extended attribute data can be represented in a single wide tuple even if the extended attributes consist of more than one `key/value` pair. The following figure shows DataDeploy's internal representation of a wide tuple. The information shown here is the same as that from the previous example. The only difference is that in this case, DataDeploy was configured to create a wide tuple.

Tuple 1
<code>path = docroot/news/front.html</code>
<code>News-Section = Sports</code>
<code>Locale = SF</code>

Wide Tuple

Notice that in a wide tuple, DataDeploy eliminates the `key =` and `value =` labels for the key and value data, instead using the format `key = value` for each `key/value` pair. This arrangement simplifies the creation of a *wide* base table as described in "Base Table Format: Wide Tuples" on page 162.

Support for wide tuples requires that all extended attribute keys be unique. For example, a file cannot have two keys named `Locale`. To satisfy this requirement, TeamXpress uses a numeric suffix for key names that would otherwise be unique. For example, if the file `docroot/news/front.html` has two `Locale` keys with the values `SF` and `Oakland`, they are named `Locale/0` and `Locale/1`. The TeamXpress GUI and metadata capture module automatically enforce this naming convention when an end user creates extended attributes for a file. The resulting wide tuple in this example is as follows:

Tuple 1
<code>path = docroot/news/front.html</code>
<code>News-Section = Sports</code>
<code>Locale/0 = SF</code>

Wide Tuple with Similar Keys

Database Object Name Lengths

To overcome the maximum database object name length imposed by database servers, DataDeploy builds a mapping table called `IWOV_IDMAPS` in the destination database. For each object name that exceeds the maximum length limit for the database, this mapping table establishes a relationship between the original object name and a generated name conforming to the database's object name length limits. The generated name is then used in place of the original object name in all database transactions. This implementation allows table names, column names, constraint names, and view names to contain any number of characters.

The `IWOV_IDMAPS` table contains three columns: `Type`, `Longid`, and `Shortid`. The `Type` column defines types as follows:

- 1: Table name
- 2: Column name
- 3: View name
- 4: Constraint name

The `Longid` column contains the entire character string for the object as it appears in the original source file. The `Shortid` column contains the generated name conforming to the database’s object length limits. For example, a typical table might appear as follows:

Typ e	Longid	Shortid
2	INFORMATION0_PRESENTATIONTITLE	IWC_AA6A93A7161
1	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING	IWT_106342E4D4C4
3	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING_VIEW	IWV_AEGF12D4E
4	INTRANET_DEPTINFO__DATADPLYBRNCH_STAGING_CONSTRAINT	IWO_F023AF1290

Because different databases support different maximum object name lengths, the threshold for when a `Shortid` name is generated depends on the database vendor and/or type. DataDeploy uses the values set for the `max-id-value` attribute to determine this threshold. See Item 10, “Database Section” in “Sample File Notes” starting on page 181 for more information. See also “Table Update Details” on page 394.

If you construct an SQL statement that performs an activity on a table that was created by DataDeploy, and if that table contains any database objects whose names exceed the maximum length, the SQL statement must first reference the mapping table to determine the actual (`Longid`) object name(s). This requirement applies to all SQL statements, including those not executed via DataDeploy.

Data Types and Sizes

The default data type for deployed data is `VARCHAR (300)`. You can set different data types, or a different size for `VARCHAR`, in the DataDeploy configuration file. See Item 11, “Rows to update” in “Sample File Notes” starting on page 181 for more information.

Incremental Deployment

DataDeploy can perform incremental deployments, in which it calculates the differences between any two specified `vpaths` and produces a delta table of the changes. The `vpaths` can be any two arbitrary TeamXpress paths such as edition paths, staging area paths, workarea paths, etc. See Item 6, “Source

Type” in “Sample File Notes” starting on page 181 for information about configuring an incremental deployment.

Deployment Scenarios

This section describes what happens when you execute a TeamXpress-to-database deployment. This type of deployment is used as an example because it is the most commonly configured deployment type, it requires the most complex configuration files, and it is the type of deployment that DAS executes.

See Appendix C, “DataDeploy Database Auto-Synchronization.” See “Sample TeamXpress-to-Database Configuration File” on page 175 for details about constructing a file to set up this type of deployment. Other deployment scenarios such as TeamXpress-to-XML, XML-to-XML, and so on, are essentially variations of the TeamXpress-to-database deployment. These scenarios are described briefly starting on page 196.

Deploying from TeamXpress to a Database: Overview

Whenever a TeamXpress-to-database deployment finishes executing, the end result is an updated table on the destination system. This table will be either a *base* table, *delta* table, or *standalone* table, depending on what type of update you instruct DataDeploy to perform (as defined in the configuration file’s `<update>` section). Update types are named for the type of table they modify. For example, a delta update modifies a delta table, and so on.

Details about each type are as follows:

- **Base update:** Extended attribute data is extracted from a TeamXpress workarea, staging area, or edition, and is deployed to a base table containing full (as opposed to delta) data about the extended attributes. The most common sources of data for a base table are staging areas and editions. Whenever a base table is generated, an entry for that table is recorded in a tracker table residing in the database. See “Data Synchronization” on page 163 for more information.
- **Delta update:** On the TeamXpress server, extended attribute data from a workarea is compared to the extended attribute data in a staging area or edition. Differences—the delta data—are identified and deployed to a delta table on the destination system. This table contains only the delta data from

the workarea; it does not contain full static data about every item in the workarea (the delta table’s associated base table should exist from a previous deployment). The relationship between the workarea data and the data in its parent area (a staging area or edition) is updated in the tracker table residing in the database. See “Data Synchronization” on page 163 for more information.

- Standalone update: Data is extracted from a TeamXpress workarea, staging area, or edition and is deployed to a standalone table containing full data about the extended attributes. A standalone update differs from a base update in that it does not generate an entry in the tracker table.

Data Sources

When you deploy extended attribute data from TeamXpress to a database, you can specify that it come from a TeamXpress workarea, staging area, or edition. Of these three, workarea data is the only type that can be deployed using any of the three types of update (base, delta, or standalone). When deploying staging area or edition data, you should use a base update if you plan subsequent delta table generation, or a standalone update if you do not need to track the table’s relationship to other tables. The following table shows which data sources are supported for each type of update:

		Update Type		
		Base	Delta	Standalone
TeamXpress Source Area	Workarea	Supported	Supported	Supported
	Staging Area	Supported	Not Supported	Supported
	Edition	Supported	Not Supported	Supported

Supported TeamXpress Source Areas for Different Types of Update

Data Destinations

In a TeamXpress-to-database deployment, the destination of deployed data can be any database on a DataDeploy server (in a three-tier system) or a database on an application server (on a two-tier system).

All tuples in all base and standalone tables will have a state of `Original`. This is because all base and standalone tuples are considered the basis against which delta tuples are compared. See “Updating a Base Table” on page 166 for an example and more details. The state of a base or standalone tuple does not reflect how or why it came to reside in the table; it simply identifies it as the basis tuple. In a delta table, the state identifies the tuple’s status relative to the same tuple in the base or standalone table.



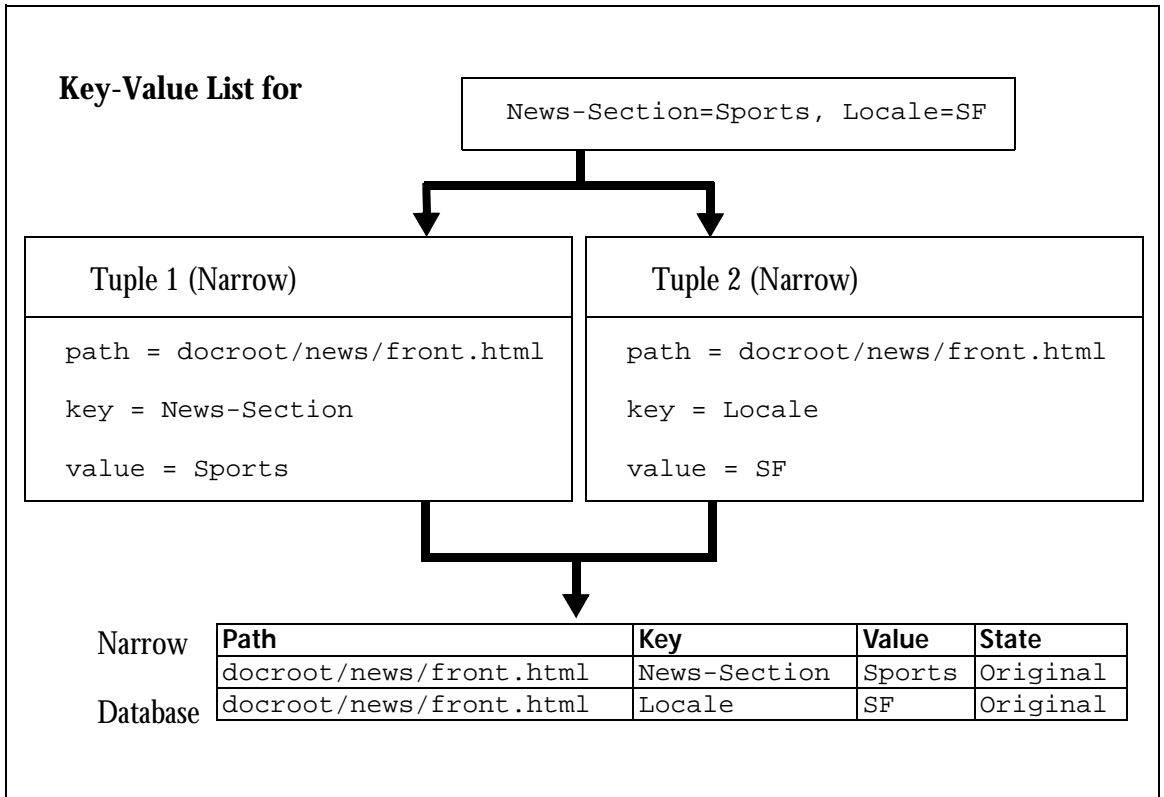
Therefore, a delta table can have tuples states of `Original`, `New`, `Modified`, or `NotPresent`. The following table shows the scenarios that can cause these states:

A delta table tuple state of:	Was caused by:
<code>Original</code>	Merging delta data from another workarea into a base table via a base update (such as when submitting the other workarea data to a staging area).
<code>New</code>	Generating a new tuple via a delta update (such as when adding a new extended attribute to a file in a workarea).
<code>Modified</code>	Updating a delta table via a delta update.
<code>NotPresent</code>	Data existing in a base area but not in a workarea (such as when the data is deleted from the workarea, or when data is newly added to the base area from a different workarea).

Delta Table Tuple States

Base Table Format: Narrow Tuples

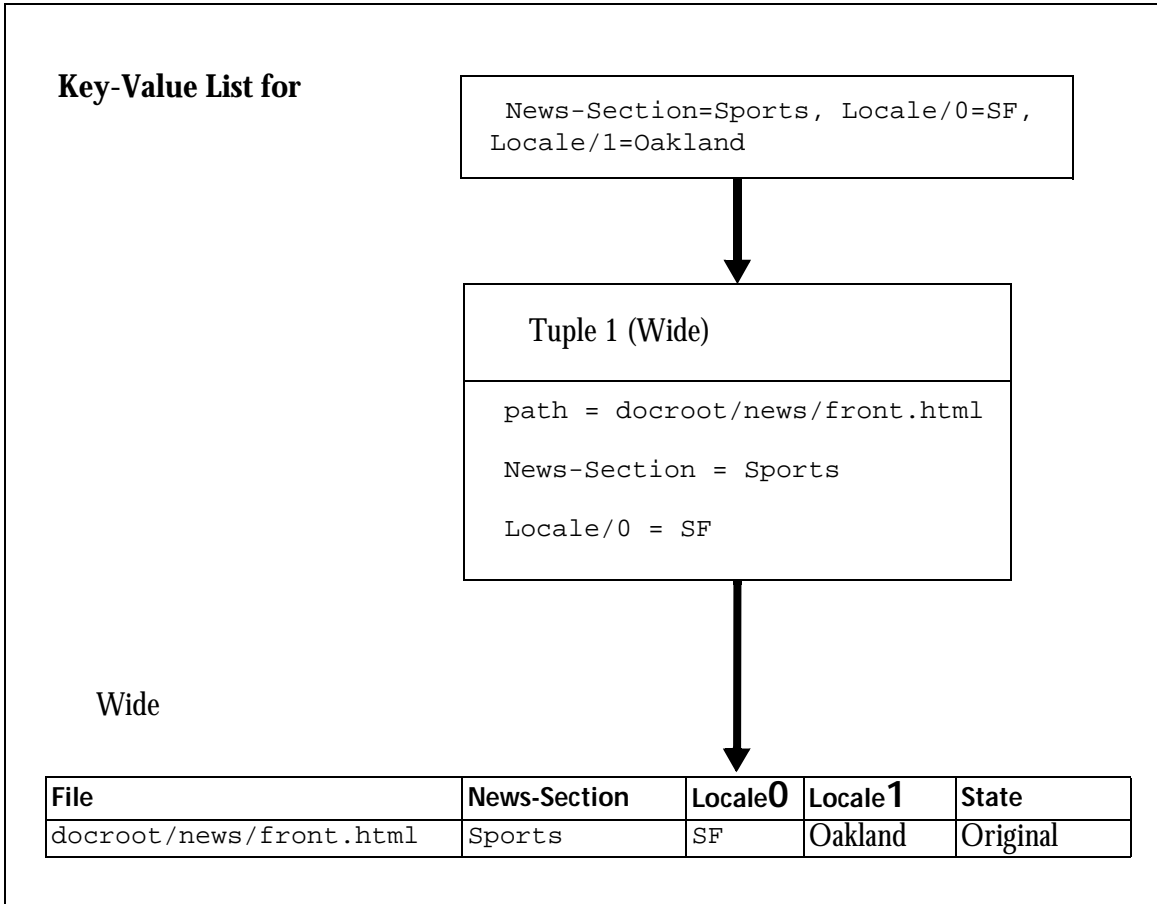
By default, deploying narrow tuples creates a base table in a database containing columns for Path, Key, Value, and State. For example:



Narrow Tuple Default Base Table

Base Table Format: Wide Tuples

By default, wide tuples deploy into wide tables, in which key values from the tuple are placed in separate columns. The end result is a table in which a single file record contains individual key value columns. For example:



Wide Tuple Default Base Table

It is also possible to deploy narrow tuples into a wide table by configuring DataDeploy to use wide tuples. When you do, the tuples are deployed to a wide table by default. See “Sample TeamXpress-to-Database Configuration File” on page 175 for guidelines about specifying wide versus narrow tuples.

You can also deploy narrow tuples to a wide table by manually configuring a set of SQL commands in the DataDeploy configuration file. These SQL commands would then execute automatically during deployment. Detailed SQL commands are beyond the scope of this document; you should refer to third party SQL documentation for more information about that topic.

Note: Table column names cannot contain reserved SQL characters such as dash (-), slash (/), question mark (?), percent (%), etc.

Data Synchronization

On the database system, DataDeploy must keep a record of which delta tables are associated with which base tables. This is necessary so that delta tables from multiple workareas that are associated with a single base table from a staging area will remain synchronized when changes from one workarea are submitted to the staging area. This relationship is maintained by the tracker table residing in the same database as the base and delta tables.

Deploying from TeamXpress to a Database: Details

The most common sequence of events when deploying from TeamXpress to a database is as follows:

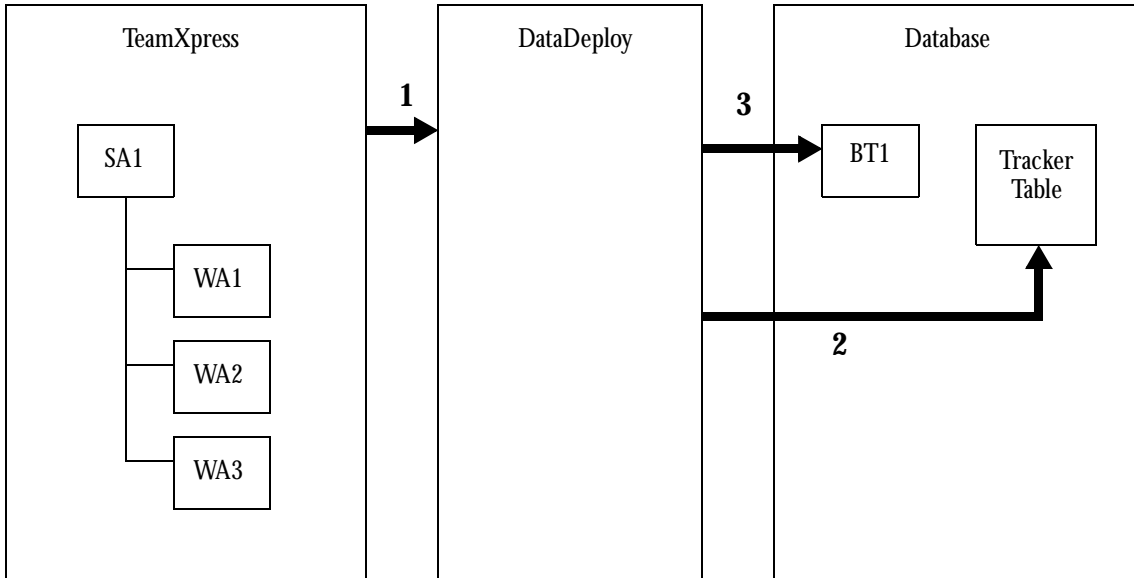
1. Generating an initial base table of a staging area or edition.
2. Generating a delta table for each workarea associated with the staging area or edition from Item 1.
3. Configuring TeamXpress to invoke DataDeploy so that the base table from Item 1 is automatically updated whenever changes are about to be submitted to its corresponding staging area or edition from a workarea.

Generating an Initial Base Table

Usually, the first action you will instruct DataDeploy to perform is the creation of an initial base table for a staging area or an edition. The following example shows the creation of a base table BT1 from a staging area SA1 on a TeamXpress branch such as `/default/main/dev/branch1`. The configuration file for this deployment is shown in “Starting-State Base Table Configuration File” on page 204. Note: In that file, the value of the attribute `name` in the `path` element is relative to the

staging area that is the source of the data being deployed. Based on the preceding conditions, the following sequence of events occurs. Refer to the figure following this list for a keyed diagram of the steps.

1. Invoke DataDeploy from the command line, specifying the deployment configuration file that contains the preceding parameters. DataDeploy reads the configuration file and goes to SA1, extracting all extended attribute data.
2. DataDeploy creates the Tracker Table (or updates it if it already exists) to track relationships between base and delta tables.
3. Based on additional information in the configuration file, DataDeploy creates base table BT1 in the destination database, populating it with the data from Step 1.

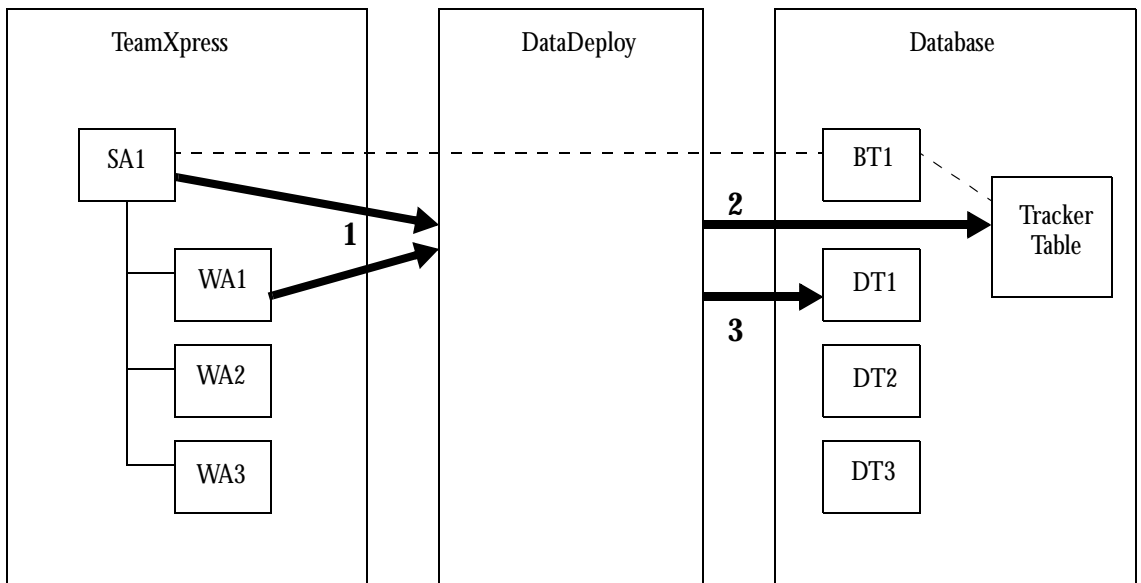


Generating an Initial Base Table

Generating a Delta Table

After creating the initial base table, you will need to generate one or more delta tables based on the workareas associated with the base table's staging area. This example shows the creation of a delta table DT1 from a workarea WA1. It assumes that a base table for SA1 has already been generated, and that WA1 is a workarea of staging area SA1. Based on the preceding conditions, the following sequence of events occurs. Refer to the figure following this list for a keyed diagram of the steps.

1. Invoke DataDeploy from the command line, specifying the deployment configuration file that contains the preceding parameters. DataDeploy compares the extended attribute data in WA1 with the same data in SA1 to determine the tuple difference between the two areas.
2. DataDeploy updates the Tracker Table to record that DT1 is a child of BT1.
3. DataDeploy creates DT1, using the delta data it determined in Step 1. If there is no delta data, it creates an empty delta table.



Generating a Delta Table

Updating a Base Table

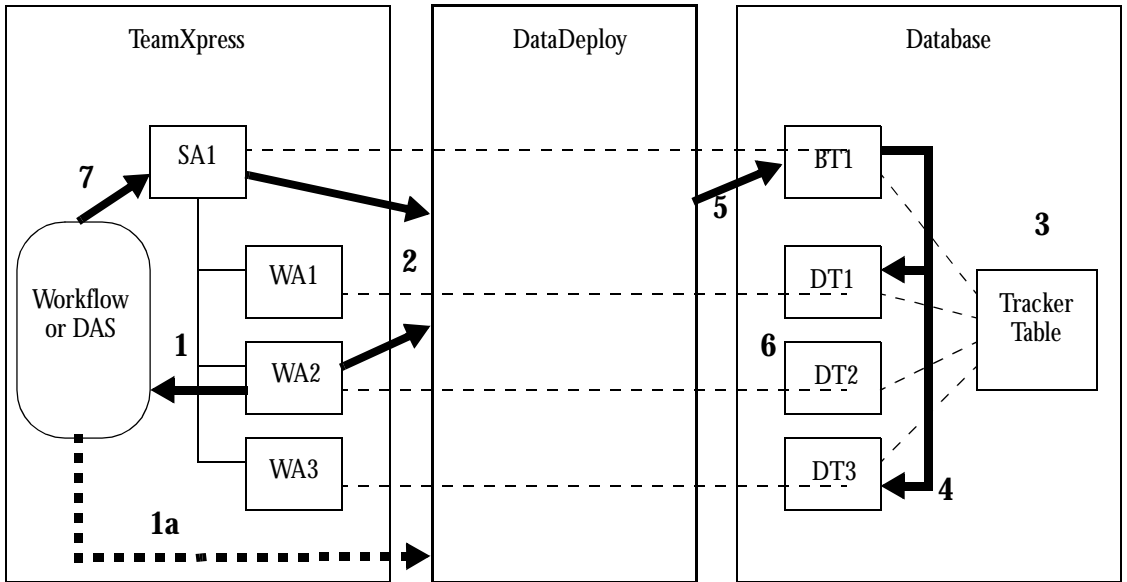
After creating the initial base and delta tables, you can configure TeamXpress workflow to automatically update a base table whenever changes in a workarea are about to be submitted to a staging area. This example assumes the following:

- You plan to submit a file list (rather than the entire workarea) from workarea WA2 to a staging area SA1.
- A base table BT1 already exists for staging area SA1.
- Delta tables DT1 through DT3 already exist for all workareas (WA1 through WA3) associated with staging area SA1.
- A tracker table already exists to establish and track the relationships between the base and delta tables.

Based on the preceding conditions, the following sequence of events occurs. Note that all of the DataDeploy activity takes place before TeamXpress actually submits the changes from WA2. Refer to the figure following this list for a keyed diagram of the steps.

1. If the submission occurs as part of a TeamXpress workflow job, the TeamXpress workflow engine obtains a list of files to be submitted from WA2 to SA1. If Database Auto-Synchronization (DAS) is configured as described in Appendix C, “DataDeploy Database Auto-Synchronization,” DAS obtains the list of files to be submitted. This list of files is then passed to DataDeploy (1a in the following figure).
2. DataDeploy compares the file list items in WA2 with the same items in SA1 to determine the tuple differences between the two areas. These differences will be installed in BT1 in Step 5.
3. DataDeploy checks the tracker table to determine the children of BT1.
4. Original rows from BT1 are propagated to DT1 and DT3 (but not to DT2). This ensures that the original rows in BT1 are not lost, but instead are stored as now-obsolete data in its child delta tables.
5. DataDeploy updates BT1 with the data derived earlier in Step 2.
6. DataDeploy removes from DT2 all rows whose path and key values are identical to those being submitted from WA2 to SA1. This ensures that items not being submitted from WA2 to SA1 are retained in DT2.

7. The workflow engine completes the submission of the file list to SA1.



Updating a Base Table

Table Updates

Hypothetical table updates for a scenario fitting this model would proceed as follows. For simplicity, the tables shown here have column headings identical to the tuple items Path, Key, Value, and State. In most situations, the columns will have other names. Because the term “key” has a specific meaning in many database languages, it is recommended that you do not use “key” as a column heading.

Starting State ¹			
BT1			
Path	Key	Value	State
P1	K1	V1	Orig
DT1			
Path	Key	Value	State

Event 1 ²			
BT1			
Path	Key	Value	State
P1	K1	V1	Orig
DT1			
Path	Key	Value	State

Event 2 ³			
BT1			
Path	Key	Value	State
P1	K1	V1	Orig
P2	K2	V2	Orig
DT1			
Path	Key	Value	State
P2	K2	V2	NtPres
DT2			
Path	Key	Value	State

Sample Table Updates

1. In their starting state, all tables are synchronized. Because there are no differences between SA1, WA1, and WA2, there is no delta data. Therefore, DT1 and DT2 are empty. This is the starting state that would exist if you completed the steps described in “Generating an Initial Base Table” on page 163. The configuration file for generating this initial version of BT1 is shown in “Starting-State Base Table Configuration File” on page 204.
2. In Event 1, workarea WA2 is changed locally with new data P2, K2, and V2, but the changes are not submitted to staging area SA1. Because the changes are not submitted, you must execute a delta update so that delta table DT2 reflects the new data in WA2. During this delta update, Data-Deploy identifies the differences between SA1 and WA2 and records these differences (the delta data) in DT2. This scenario is similar to the scenario in “Generating a Delta Table” on page 165. However, in that scenario a delta table did not exist yet and had to be generated for the first time.

In the scenario shown here, the delta tables already exist and therefore only need to be updated. The configuration file for this delta deployment is shown in “Event 1 Configuration File” on page 205.

3. In Event 2, workarea WA2 (complete with its changes from Event 1) is submitted to staging area SA1. In the configuration file for this deployment, Path and Key were named as the basis-for-comparison columns. Therefore, DataDeploy compares the Event 1 values of these columns in BT1 and DT2, sees that they are different, and determines that the row from DT2 Event 1 should append rather than replace the data in BT1. DT1 has the new values shown here because WA1 now differs from SA1. If necessary, a Get Latest operation in WA1 would bring WA1 into sync with SA1. (Had Event 1 DT2 contained a P1 K1 V2 row, it would have replaced rather than appended the original BT1 row. In that case, the original BT1 row would have been propagated to DT1, after which P1 K1 V2 would have replaced P1 K1 V1 in BT1. A subsequent Get Latest in WA1 would bring WA1 into sync with SA1, and the data in DT1 would be deleted). DT2 is empty because WA1 is once again in sync with SA1. This is the ending state that would exist if you completed the steps described in “Updating a Base Table” on page 166. The configuration file for this deployment is shown in “Event 2 Configuration File” on page 206. Note: In that file, all of the items in `filelist` are path-relative to `area`.

Composite Table Views

There are three ways that you can create table views:

- Through SQL commands that you execute manually to query the database after it is created. See “DataDeploy Querying Tables” on page 407 for more information.
- Through SQL commands named in the `user-action` attribute of the DataDeploy configuration file’s `<sql>` element. You run these commands by executing an SQL-specific deployment that you specify via the command line options `iwdd-op=do-sql` and `user-op=anyname`. See “Sample File Notes” on page 181 and “Invoking DataDeploy” on page 207 for more information.
- By setting the `table-view` attribute in the DataDeploy configuration file’s `<database>` section. See “Sample File Notes” on page 181 for more information.

The following composite views for workareas WA1 and WA2 would result from the scenarios described in the previous sections. The composite for WA1 is the result of querying BT1 and DT1



using the SQL statements described in “DataDeploy Querying Tables” on page 407. Likewise, the composite for WA2 is the result of querying BT1 and DT2.

Starting State			
WA1			
Path	Key	Value	State
P1	K1	V1	Orig
WA2			
Path	Key	Value	State
P1	K1	V1	Orig

Event 1			
WA1			
Path	Key	Value	State
P1	K1	V1	Orig
WA2			
Path	Key	Value	State
P1	K1	V1	Orig
P2	K2	V2	New

Event 2			
WA1			
Path	Key	Value	State
P1	K1	V1	Orig
WA2			
Path	Key	Value	State
P1	K1	V1	Orig
P2	K2	V2	Orig

Composite Table Views

Configuration File Details and Examples

This chapter contains the following detailed information about configuration file contents:

- Which elements are required in each type of configuration file.
- Rules for parameter substitutions within configuration files.
- An annotated sample TeamXpress-to-database configuration file.
- A sample TeamXpress-to-XML configuration file.
- A sample database-to-database configuration file.
- A sample database-to-XML configuration file.
- A sample XML-to-database configuration file.
- A sample XML-to-XML configuration file.
- The configuration files for “Starting State,” “Event 1,” and “Event 2” shown on page 168.

Required Elements

The type of deployment (e.g., TeamXpress-to-database, TeamXpress-to-XML, and so on) determines which configuration file sections are required and which elements can reside in each section. Only a few parameters are actually required within these sections. The rest are optional, making it possible to have short, simple configuration files. Section hierarchy and requirements for each supported type of deployment are as follows. Sections in bold text are required; those in normal text are optional. Indentation shows nesting levels.

TeamXpress-to-Database

```
filter
  keep
  discard
substitutions
data-deploy-elements
client
deployment
  substitutions
  exec-deployment
  source
    TeamSite-extended-attributes
    TeamSite-templating-records
  destinations
    substitutions
    filter
  database
    select
    update
    sql
server
```

TeamXpress-to-XML

```
filter
  keep
  discard
substitutions
client
data-deploy-elements
deployment
  substitutions
  exec-deployment
  source
    TeamSite-extended-attributes
    TeamSite-templating-records
  destinations
    substitutions
    filter
  xml-formatted-data
server
```

Database-to-Database

- filter
 - keep
 - discard
- substitutions
- data-deploy-elements
- client
- deployment**
 - substitutions
 - exec-deployment
 - source**
 - database**
 - fields
 - destinations
 - substitutions
 - filter
 - database**
 - select
 - update
 - sql
- server

Database-to-XML

- filter
 - keep
 - discard
- substitutions
- data-deploy-elements
- client
- deployment**
 - substitutions
 - exec-deployment
 - source**
 - database
 - fields
 - destinations
 - substitutions
 - filter
 - xml-formatted-data**
- server

XML-to-Database

```
filter
  keep
  discard
substitutions
client
data-deploy-elements
deployment
  substitutions
  exec-deployment
  source
    xml-formatted-data
      fields
destinations
  substitutions
  filter
  database
    select
    update
    sql
server
```

XML-to-XML

```
filter
  keep
  discard
substitutions
data-deploy-elements
client
deployment
  substitutions
  exec-deployment
  source
    xml-formatted-data
      fields
destinations
  substitutions
  filter
  xml-formatted-data
server
```


Parameter Substitutions

Any parameter string in a configuration file can be named using a parameter substitution. You set parameter string substitutions on the same command line you use to invoke DataDeploy with the `iwdd` command. Syntax is as follows:

```
"varname=varvalue"
```

After a string is defined on the command line, all occurrences of `$varname` in the configuration file named on the command line are substituted with the string `varvalue`. Do not use the following terms for `varname`; they are keywords for the `iwdd` command and would be interpreted as such:

```
cfg
deployment
iwdd-op
remote-host
remote-port
```

Examples of parameter substitution within a configuration file are as follows:

```
prefix_string_$varname
$varname^_suffix_string (where ^ is a concatenator)
prefix_$varname^_suffix
```

Sample TeamXpress-to-Database Configuration File

The following sample configuration file shows how to set parameters for a typical TeamXpress-to-database deployment. It identifies which parameters are required, shows both global and in-flow usage, and is keyed to a comment table following the file that explains more details about each section and parameter. Most of the elements in this file are also used to define types of deployment other than TeamXpress-to-database. For examples of configuration files for these other deployment types, see the sample file sections starting on page 196.



```

<!--Sample DataDeploy configuration file -->
<data-deploy-configuration>
  <data-deploy-elements filepath="/local/iw-home/db.xml"/>
  <filter name="MyFilter">
    <!-- This is a filter that can be used by any deployment -->
    <keep>
      <!-- Any of the following (logical OR): -->
      <!-- dir2/subdir/index.html, any *.html file in dir1, -->
      <!-- OR anything with key 'guard' AND value 'IGNORE -->
      <or>
        <field name="path" match="dir2/subdir/index.html" />
        <field name="path" match="dir1/*.html" />
      <and>
        <!-- Must match all of these (logical AND) -->
        <field name="key" match="guard" />
        <field name="value" match="IGNORE" />
      </and>
    </or>
  </keep>

  <discard>
    <!-- Exclude the file dir1/ignoreme.html, anything -->
    <!-- with key 'unneededKey', and anything with state -->
    <!-- DELETED -->
    <or>
      <field name="path" match="dir1/ignoreme.html" />
      <field name="key" match="unneededKey" />
      <field name="state" match="DELETED" />
    </or>
  </discard>
</filter>

<substitution name="GlobalSubstitution">
  <!-- This substitution can be used by any deployment. -->
  <!-- It replaces the first occurrence of the string 'foo' -->
  <!-- in the 'path' field with 'bar', and completely -->
  <!-- replaces the 'value' field with the string 'SpecialValue'.-->
  <field name="path" match="foo" replace="bar" />
  <field name="value" replace="SpecialValue" />
</substitution>

```

Include file ¹

Filter section (global) ²

Substitution section (global) ³

```

<client>
  <!-- This deployment puts EA data from a TeamXpress area into -->
  <!-- a destination database. -->
  <deployment name="ea-to-db">
    <source>
      <!-- Pull data tuples from (local) TeamXpress EA's. -->
      <!-- Only those EA's that are different from the -->
      <!-- ones in the base area will be reported. The -->
      <!-- actual workarea will be taken from the 'user' -->
      <!-- command-line parameter. -->
      <TeamSite-extended-attributes
        options="differential, wide"
        area="/default/main/dev/branchx/WORKAREA/$user"
        base-area="/default/main/dev/branchx/STAGING">
        <path name="dir1/index.html" visit-directory="no" />
        <path name="dir2/subdir" visit-directory="shallow" />

        <!-- Use the command-line parameter 'path' -->
        <!-- as the path name. If the path happens -->
        <!-- to be a directory, visit its children -->
        <!-- recursively. -->
        <path name="$path" visit-directory="deep" />

        <!-- Read a list of files from the file -->
        <!-- '/tmp/SomeFiles'. The default directory -->
        <!-- mode 'deep' will be used for each file. -->
        <path filelist="/tmp/SomeFiles" />
      </TeamSite-extended-attributes>
    </source>

    <!-- Apply global filter 'MyFilter' to all tuples -->
    <filter use="MyFilter" />
  </deployment>
</client>

```

Start of Client section⁴

Data type⁷ (required)

End of Source section⁶ (required)

Start of Deployment section⁵ (required)

Start of Source section⁶ (required)

Location of source data⁸ (area)

Call global filter²



```

<substitution>
  <!-- Modify each tuple according to the following -->
  <!-- match/replace pairs. In this case: any path -->
  <!-- that contains the string 'WORKAREA/.../' will -->
  <!-- have the string replaced by 'STAGING/'; any -->
  <!-- path that contains 'EDITION/abcd' will be -->
  <!-- replaced with '/This/Special/Path', and any -->
  <!-- tuple whose key starts with 'BEFORE' will be -->
  <!-- changed to begin with 'AFTER'. -->
  <field name="path"
    match="(.*)/WORKAREA/[^/]+/(.*)"
    replace="\1/STAGING/\2" />
  <field name="path"
    match="EDITION/abcd"
    replace="/This/Special/Path" />
  <field name="key"
    match="^BEFORE(.*)"
    replace="AFTER\1" />
</substitution>
<!-- Also apply the substitution 'GlobalSubstitution' -->
<substitution use="GlobalSubstitution" />

```

Substitution section (in-flow) ⁹

← Call global substitution ³

```

<!-- Start the destinations section. -->

```

```

<destinations
  host="DDServer.interwoven.com"
  port="1357">
  <!-- Filtered and substituted data will be sent to a -->
  <!-- DataDeploy server on port 1357 of the machine -->
  <!-- DDServer.interwoven.com. Then -->
  <!-- send some tuples to 'table1' on the database that -->
  <!-- is located using 'jdbc:remote.machine.com' and -->
  <!-- provide user 'dba' with password 'ThisIsASecret'. -->
  <!-- Perform any other activities that are associated -->
  <!-- with the option 'ea-update'. Timeout is 45 secs. -->
  <database name="myproductiondb"
    db="host1:1357:db1"
    table="table1"
    user="dba"
    password="ThisIsASecret"
    timeout="45">

```

Start of Destinations section ¹⁰ (required)

Start of Database section and location of destination database ¹¹ (required)

```
Rows to  
update 12  
(required)  <select>  
              <!-- Select the row whose value in the column  -->  
              <!-- named 'filename' matches the current path, -->  
              <!-- whose value in column 'InterestingTag'  -->  
              <!-- matches the current key as modified by any -->  
              <!-- substitutions, and that has literal      -->  
              <!-- value 'litData' in column 'info'.        -->  
              <column name="filename"  
                value-from-field="path" />  
              <column name="InterestingTag"  
                value-from-field="key" />  
              <column name="info"  
                value="litData" />  
            </select>  
  
Update type  
and related  
data 13  
(required)  <update type="delta"  
              base-table="RootTable1"  
              state-field="StateInfo">  
              <!-- Update column 'RelatedValue' to contain the -->  
              <!-- current EA value, and update the column      -->  
              <!-- whose name is taken from the 'key' field    -->  
              <!-- with the literal value 'present'. The table -->  
              <!-- being updated is assumed to be a delta      -->  
              <!-- table modifying base table 'RootTable1';   -->  
              <!-- the differencing operations are driven by  -->  
              <!-- the value of tuplefield 'StateInfo'.        -->  
              <column name="RelatedValue"  
                value-from-field="value" />  
              <column name-from-field="key"  
                value="present" />  
            </update>
```

Columns to
update ¹⁴
(required)



```

SQL
section 15
<!-- If it is necessary to create a new table for -->
<!-- this deployment, the following SQL statement -->
<!-- will be used for that purpose (as opposed to -->
<!-- a capriciously chosen internal default) -->
<sql action="create">
  <!-- This comment should be ignored. However -->
  <!-- the parameter token in the next line is -->
  <!-- subject to parameter substitution. -->
    CREATE TABLE table1 (
      Path VARCHAR(300) NOT NULL,
      KeyName VARCHAR(300) NOT NULL,
      Value VARCHAR(4000) ,
      State VARCHAR(4000) ,
      CONSTRAINT KVP PRIMARY KEY (Path,KeyName)
    )
  </sql>
</database>
</destinations>
</deployment>

</client>

<server>
  <!-- The DataDeploy server will listen on port 1949 of IP -->
  <!-- 204.247.118.99 -->
  <bind ip="204.247.118.99" port="1949" />

  <!-- Only accept connections from these hosts -->
  <allowed-hosts>
    <host addr="ddclient.interwoven.com" />
    <host addr="204.247.118.33" />
  </allowed-hosts>

  <!-- Server-specific deployment information -->
  <for-deployment name="ea-to-db">
    <database db="host1:1357:db1"
      user="scott"
      password="tiger" />
  </for-deployment>
</server>
</data-deploy-configuration>
Server section 16

```

Sample File Notes

- 1. Include File:** You can use `<data-deploy-elements>` to name a file containing data to include by reference. The file named in `<data-deploy-elements>` can contain any number of `<database>`, `<filter>`, and `<substitution>` elements. It must use the same syntax for these elements that the main DataDeploy configuration file uses. See Items 2, 3, and 11 in this section for details. See page 227 for a complete sample include file. If mutually exclusive attributes are set in the include file and the main DataDeploy configuration file, all are used in the deployment. If conflicting attributes are set in the two files, those set in the main DataDeploy configuration file take precedence.
- 2. Filter section (global):** Filters let you explicitly state which tuples will or will not be deployed. The `keep` section contains criteria for selecting which tuples are deployed, and the `discard` section contains criteria for those which are not. Both sections use `field` tags. All `field` tags must contain at least one `name/match` attribute pair. When you deploy from TeamXpress, `name` must be either `key`, `value`, `path`, or `state` (as defined earlier in “Data Organization Within DataDeploy” on page 154). When you deploy from a source other than TeamXpress, `name` can be any be any field name that is valid in the source area. The `match` attribute names a targeted value for `name`. A filter defined in the nesting level shown here and located before the Deployment section will be global. Global filters do not become active until they are called via the `<filter>` element’s `use` attribute between the Source and Destinations sections using the syntax shown later in the sample file. Note that filters can also be defined in an include file and then be called via the `use` attribute. If a configuration file does not contain a filter section, all tuples are deployed (limited only by the type of update being performed). A configuration file can contain any number of global filter sections. A configuration file can also contain in-flow filters within a `destinations` section. See Item 10 for details.
- 3. Substitution section (global):** Substitutions let you configure DataDeploy to automatically replace character strings or entire fields in a table. Substitutions use `field` tags that must contain at least one `name/replace` attribute pair. As with filters, `name` is either `key`, `value`, `path`, or `state`. The `replace` attribute is the new string that will overwrite the existing string or field. Two additional attributes, `match` and `global`, are optional. Common usage examples are as follows:



To do this:	Include this line in the Substitution section:
Replace all Value field values with the string <code>Newvalue</code>	<code><field name="value" replace="Newvalue" /></code>
In the Path field, replace first occurrence of <code>blue</code> with <code>red</code>	<code><field name="path" match="blue" replace="red" /></code>
In the Path field, replace all occurrences of <code>blue</code> with <code>red</code>	<code><field name="path" match="blue" replace="red" global="yes" /></code>
In the State field, replace the first occurrence of <code>Original</code> with <code>NotPresent</code>	<code><field name="key" match="Original" replace="NotPresent" /></code>

A substitution defined in the nesting level shown here and located before the Deployment section will be global. Global substitutions do not become active until they are called via the `<substitution>` element's `use` attribute between the Source and Destinations sections using the syntax shown later in the sample file. Note that substitutions can also be defined in an include file and then be called via the `use` attribute. A configuration file can contain any number of global substitution sections.

- Client section:** The `client` section lets you specify a set of client-specific parameters and activities. A configuration file that is expected to run on a two-tier system or as a client on a three-tier system must have exactly one client section.
- Deployment section:** The `deployment` section is where you assign a name to each deployment, and specify deployment source, destination, and update type. You can have any number of `deployment` sections in a configuration file, and each must have a unique name. The name shown here, `ea-to-db`, is the name you would specify on the command line when you invoke `DataDeploy`. The `deployment` section is required in all configuration files. The `<exec-deployment>` subelement lets you execute one or more deployments that are defined elsewhere in the same configuration file. Syntax is as follows:

```
<exec-deployment use="dbname" />
```

where `dbname` refers to the name of a database as defined in the `name` attribute in a `<database>` element.

6. **Source section:** The `source` section resides one nesting level inside the `deployment` section. It is where you name the type of data to extract from TeamXpress and the location(s) of that data. Each `deployment` section must have exactly one `source` section.
7. **Source type:** The first nesting level within the `<source>` element contains a subelement defining what type of data is to be extracted from TeamXpress. This subelement has the following possible elements:

Subelement	Description
<code>TeamSite-templating-records</code>	Used when deploying a TeamXpress Templating data content record from TeamXpress. Supported options: <code>wide</code> (default), <code>full</code> (default), <code>differential</code> .
<code>TeamSite-extended-attributes</code>	Used when deploying anything other than a data content record from TeamXpress. Supported options: <code>narrow</code> (default), <code>wide</code> , <code>full</code> (default), <code>differential</code> .
<code>xml-formatted-data</code>	Used when deploying from an XML file. Supported options: <code>narrow</code> (default), <code>wide</code> , <code>full</code> (default), <code>differential</code> .
<code>database</code>	Used when deploying from a database. Supported options: <code>narrow</code> (default), <code>wide</code> , <code>full</code> (default), <code>differential</code> .

Each of the preceding subelements supports three attributes: `options`, `area`, and `base-area`. Details about the `options` attribute are as follows:

options Value	Description
<code>wide</code>	Creates a wide table based on wide tuples containing any number of key/value pairs. Specified in addition to <code>differential</code> or <code>full</code> . The <code>wide</code> and <code>narrow</code> values are mutually exclusive; you cannot specify both within the same element. The <code>wide</code> value is the default for the <code>TeamSite-templating-records</code> element.



narrow	Creates a 4-column (narrow) table based on narrow tuples. Specified in addition to <code>differential</code> or <code>full</code> . The <code>wide</code> and <code>narrow</code> values are mutually exclusive; you cannot specify both within the same element. The <code>narrow</code> value is the default for the <code>TeamSite-extended-attributes</code> , <code>xml-formatted-data</code> , and <code>database</code> elements. The <code>TeamSite-templating-records</code> element does not support the <code>narrow</code> value.
differential	Instructs <code>DataDeploy</code> to extract just the delta data from a workarea/staging-area comparison. Normally, you specify <code>differential</code> when performing a delta update. The <code>differential</code> and <code>full</code> values are mutually exclusive; you cannot specify both within the same element. The default is <code>full</code> .
full	Instructs <code>DataDeploy</code> to create a table populated with all of the data from a named area. Normally, you specify <code>full</code> when performing a base or standalone update (update types are defined later in the <code>destinations</code> section). The <code>differential</code> and <code>full</code> values are mutually exclusive; you cannot name both as options within the same element. The default is <code>full</code> .

To configure an incremental deployment, set the `<TeamSite-extended-attributes>` or `<TeamSite-templating-records>` elements as follows. The result is a delta table containing the differences between `vpath1` and `vpath2`.

```
<TeamSite-extended-attributes
  options="differential"
  area="vpath1"
  base-area="vpath2"

  ...additional subelements if necessary...

</TeamSite-extended-attributes>
```

- 8. Location of source data:** The `area` attribute defines the TeamXpress workarea, staging area, or edition from which `DataDeploy` will extract data. This attribute is required in all deployment sections. The value of `area` should be the `vpath` name of the area containing the changes you intend to deploy. If `differential` is set, you must also supply a `vpath` value for `base-area`. This value should be the `vpath` name of the edition or staging area that is the basis for comparison with the workarea you named in `area`. The optional `path` element can have one (but not both) of the following values: `name` or `filelist`. Setting the `name` attribute lets you specify a relative path name to a file or directory in the area(s) you named earlier in `area` (and `base-area` if applicable), or stipulate that the path name will be entered

on the command line when you invoke DataDeploy. See “Parameter Substitutions” on page 175 for information about entering path names on the command line. Setting the `filelist` attribute lets you specify a file containing a list of files, and is typically used when you perform a delta update of a workarea containing only a few changed files. If you do not name a `path` value, it defaults to “.” and DataDeploy performs a deep search of the directory named in `area` (and `base-area` if applicable). The `visit-directory` attribute lets you specify DataDeploy’s level of searching within a directory. The three possible values are `no`, `shallow`, and `deep`. Details are as follows:

Value	Description
<code>no</code>	If <code>path</code> name is a directory, it is not searched.
<code>shallow</code>	If <code>path</code> name is a directory, it is searched to the first level.
<code>deep</code>	All directories and all subdirectories found in <code>path</code> name are searched recursively.

The default value of `visit-directory` is `deep`.

- 9. Substitution section (in-flow):** In-flow substitutions let you define substitution rules that apply only to specific parts of a deployment. DataDeploy supports in-flow substitutions within the `deployment` and `destinations` elements. For example, the in-flow substitution shown in the sample configuration file is nested one level inside of the `deployment` element, and therefore applies only to the `ea-to-db` deployment. You can also nest in-flow substitutions one level inside `destinations` elements, in which case the substitution applies only to a specific destination. In-flow substitutions have the same syntax as global substitutions. In addition, in-flow substitutions support a `global` attribute that lets you that lets you control whether the substitution applies to all occurrences or just the first occurrence of the matching pattern.



If `global` is set to `no`, the substitution applies only to the first occurrence. If it is set to `yes`, the substitution applies to all occurrences. For example:

```
<destinations>
  <database . . .>
  <substitution name="SubForThisTarget">
    <field name="BField" match="from_a"
      replace="to_b"
      global="yes" />
  </substitution>
```

The example shown in the sample configuration file earlier in this chapter uses Perl 5 regular expression syntax for `match` values. A configuration file can contain any number of in-flow substitution sections.

- 10. Destinations section:** The `destinations` section resides one nesting level inside the `deployment` section. It is where you name the destination system(s), timeout value, database, and table, and is also where you define the update type. Each `deployment` section can have any number of `destinations` sections, allowing you to designate multiple destinations in a single configuration file. Destination system and timeout details are as follows. Database, table, and update type are explained later in Item 11.

Attribute	Description	Required?	Value Syntax
host	Machine name of the DataDeploy server (3-tier systems only).	No	"hostname.com"
port	Port on <code>host</code> to which data will be sent.	No	"portnumber"
timeout	How long the client system will wait for a response from the remote host during communication exchange. This tag can also reside in the Database section, in which case it has a different definition. See Item 11 for details.	No	"seconds"

You can also nest in-flow filters within a `destinations` element, in which case the filter applies only to that specific destination. For example:

```
<destinations>
  <database . . .>
  <filter name="FilterForThisTarget">
```

```

    <discard>
      <field name="AField" match="^DoNotWant/.*/>
    </discard>
  </filter>

```

In-flow filters have the same syntax as global filters.

11. Database section: The first subelement in the `destinations` section defines the type of destination for the data. This subelement can be either `<database>` or `<xml-formatted-data>`, depending on whether the destination is a database or an XML file. See “Sample TeamXpress-to-XML Configuration File” on page 196 for an example of `xml-formatted-data` usage. When deployment is to a database, the `<database>` tag and its `name` and `db` attributes are required in all `deployment` sections. A `destinations` section can have any number of `<database>` subelements or a combination of `<database>` and `<xml-formatted-data>` subelements. Syntax for the values `db` and other attributes of the `<database>` tag are as follows:

Attribute	Value	Description	Required ?
<code>name</code>	Any user-defined database name surrounded by double quotes, e.g., <code>"myproductiondb"</code> .	Used to reference the database via the <code>use</code> attribute elsewhere in the configuration file. For example, the <code><exec-database></code> element could contain <code>use="myproductiondb"</code> .	Only if the <code>use</code> attribute is used elsewhere in the file.
<code>db</code>	Depends on vendor; see next table.	Names the address string of the destination database.	Yes.*
<code>use</code>	The name of the database set by the <code>name</code> attribute.	If a database is defined in an include file, you can reference it as a destination by including it here. If you reference a database via <code>use</code> , you do not need to specify <code>name</code> or <code>db</code> in the reference because they are already defined in the include file. However, you can optionally set <code>db</code> or any other attribute together with the <code>use</code> attribute, in which case the explicitly set attributes take precedence.	No.



Attribute	Value	Description	Required ?
table	Any user-defined table name surrounded by double quotes, e.g., "table1".	Names a destination table in db.	Yes.*
user	Any user name surrounded by double quotes, e.g., "user1".	Authorizes a specific database user.	Yes.*
password	Any user-defined password surrounded by double quotes, e.g., "w21YS".	Names the assigned password for user. Note that any password named in a configuration file is not encrypted, and can be read by anyone having access to the configuration file.	Yes.*
timeout	Any positive integer representing the duration of the timeout in seconds, surrounded by double quotes, e.g., "4".	How long the client system will attempt to log into the database system before giving up. This tag can also reside in the Destinations section prior to the Database section, in which case it has a different definition. See Item 10 for details.	No.
clear-table	"yes" or "no"	Specifies whether a delta table should be cleared before receiving new data. Useful to set to yes (which is the default) when deleting many workarea files prior to submitting. Set to no if updating extended attributes on existing files prior to submitting.	No.
table-view	"yes" or "no"	Specifies whether to create a view automatically during deployment. The default is no. Setting to yes is incompatible with Sybase ASE (but works correctly with Sybase SQLAnywhere and all other supported databases). Setting to yes and using Sybase ASE will result in an aborted deployment.	No.

Attribute	Value	Description	Required ?
vendor	"microsoft"	Specifies Microsoft SQLServer. Sets a default <code>max-id-length</code> of 128.	Yes.
	"oracle"	Specifies an Oracle database. Sets a default <code>max-id-length</code> of 30.	
	"sybase"	Specifies Sybase SQLAnywhere. Sets a default <code>max-id-length</code> of 128.	
	"ibm"	Specifies IBM DB2. Sets a default <code>max-id-length</code> of 30.	
	"informix"	Specifies an Informix database. Sets a default <code>max-id-length</code> of 18.	
max-id-length	Any positive integer appropriate for an object name length (per the documentation provided by the database vendor).	Specifies the maximum number of characters in any database object name (e.g., column names, table names, etc.), overriding any defaults set via the <code>vendor</code> attribute.	No.**

* Either here or in the Server section's Database section. See Item 16.

**Not required, but highly recommended. Even if the appropriate value is set via the `vendor` default, setting it again in `max-id-length` ensures that the value is explicitly set and easily verified. This also ensures that the value will remain constant should the default value (as set dynamically by DataDeploy) ever change.

The syntax for the value of the `db` attribute shown in the preceding table depends on the database vendor. Details are as follows. Syntax and example lines should all be on one line in the DataDeploy configuration file. Line breaks shown here are due to formatting constraints of this document.

Database/Driver	Syntax of db Attribute	Example
Informix	<code>db="//host_name:port/database_name:INFORMIXSERVER=server_name"</code>	<code>db="//sys1.interwoven.com:1357/bank01:INFORMIXSERVER=OL_sys1"</code>
Oracle/JDBC thin	<code>db="host_name:port:instance_identifier"</code>	<code>db="host1:1357:db1"</code>



Database/Driver	Syntax of db Attribute	Example
Oracle/JDBC OCI *	db="database_tnsname "	db="bank01 " (See Oracle documentation for details about configuring TNS names)
Sybase SQL Anywhere	db="JDBC_data_source_name "	db="server1 "
Sybase ASE	db="host_name:port/database_name "	db="sys1.interwoven.com:1357/bank01 "
Microsoft SQL Server	db="data_source_name "	db="bank01 " (See Microsoft documentation for details about creating data source names on Windows NT and Windows 2000 systems)
IBM DB2 (UDB)	db="//host_name:port/database_name "	db="//host1:1357/db1 "

* Used by DataDeploy if Oracle extension datatypes (e.g., CLOB) are used. Requires installation of the OCI client library on the system from which the `iwdd.ip1` command is executed.

The `<database>` subelement also supports the `<stored-procedure>` subelement, which allows you to deploy key-value pairs that are treated as a stored procedure. The `<stored-procedure>` subelement resides in the first nesting level within the `<database>` element, and lets you write a stored procedure using standard SQL syntax as supported by the current database. You can then store the procedure in the database by deploying it as an extended attribute via DataDeploy. Syntax is as follows:

```

<stored-procedure>
  <fieldname prefix="any_prefix_1"/>
  <fieldname prefix="any_prefix_2"/>
  <fieldname prefix="any_prefix_n"/>
</stored-procedure>

```

The value of `any_prefix` can be any case-insensitive character string. DataDeploy will examine each tuple for key-value pairs in which the key name starts with any of the specified prefix values.

For each match, the value for that key is treated like a database stored procedure; that is, DataDeploy does not validate the value of the key-value pair for syntax and semantic correctness. Instead, DataDeploy passes the value to the database, the key-value pair is not inserted into the table, and errors (if any) are returned to the user. If creation of stored procedure fails and if the tuple contains non-stored procedure key-value pairs, the entire deployment is aborted.

12. Rows to update: The `select` section is where you select database rows to update with data from the current tuple. It is also where you can specify a data type for the deployed data other than the default `VARCHAR 300` (you can also set the data type in Update section; see Item 13, “Update type and related data”). You identify rows by stating one or more matching criteria for column values in that row. For example, you can select a row whose values in columns named “color” and “size” are respectively “red” and “small.” Column matching criteria are set through the `column` tag. Each database section must have exactly one `select` section, and each `select` section must contain at least one `column` tag. Each column tag must contain the following two attributes:

- 1) `name` OR `name-from-field`
- 2) `value` OR `value-from-field`

The column tag can optionally contain the `data-type` and `data-format` attributes.

Syntax is as follows:

Attribute	Description	Value Syntax
<code>name</code>	Specifies a column by name.	Text string containing any column name from the table specified by the <code>database</code> tag.
<code>name-from-field</code>	Specifies a column name by reference to a field in the current tuple.	Any of the following: <code>key</code> , <code>value</code> , <code>path</code> , <code>state</code> .
<code>value</code>	Specifies the literal value to match in the column just named.	Text string containing any table value.
<code>value-from-field</code>	Specifies a value to match by reference to a field in the current tuple.	Any of the following: <code>key</code> , <code>value</code> , <code>path</code> , <code>state</code> .



data-type	Specifies the datatype for the extended attributes being deployed. If not set, DataDeploy assumes a data type of VARCHAR.	Any datatype supported by the database.
data-format	<p>Only required if data-type is set. Specifies the format of the extended attributes being deployed as a date or time. Can be used only under the following conditions:</p> <ul style="list-style-type: none"> On an Oracle database server, and when data-type is either DATE, DATETIME, or TIMESTAMP. If you set data-format when any of these conditions do not exist, the setting is ignored. If data-type is either DATE or DATETIME. The format of the data-format value must conform to the specification described for the SimpleDateFormat Java class. 	Any valid date or time format.

For example, you would use the following <column> element configuration to deploy the KeyName1 extended attribute values as integers:

```
<column name="ValueCol"
      data-type="INT"
      value-from-field="KeyName1" />
```

Or, to deploy KeyName1 extended attribute values as a date formatted to show Year-Month-Day Hours:Minutes:Seconds (assuming an Oracle database):

```
<column name="ValueCol"
      data-type="DATE"
      data-format="YYYY-MM-DD HH24:MI:SS"
      value-from-field="KeyName1" />
```

If the data-type attribute is not specified in the DataDeploy configuration file, DataDeploy uses VARCHAR (300) as the datatype. If a large number of columns are created in the table, the total size of each row could easily exceed the maximum row size imposed by the database server.

Therefore, it is recommended that you set the `data-type` attribute whenever possible for the columns defined in the `<select>` and `<update>` sections of the DataDeploy configuration file.

- 13. Update type and related data:** The `update` section is where you select the type of update, reference table (if applicable), and the table column(s) to update. Update type can be `delta`, `base`, or `standalone` (the default). Type `delta` requires two attributes, `base-table` and `state-field`. The `base-table` attribute names the base table that will be modified after the delta table (named earlier in the `database` section) is updated. The `state-field` attribute names which tuple item will be interpreted as state information. Each `database` section must have exactly one `update` section. The relationship between `update` section settings and the table named earlier in the `database` section's `table` attribute is as follows:

If the Update section contains this:	And the Database section contains this:	The result is:
<code>type = "base"</code>	<code>table="Table1"</code>	DataDeploy assumes <code>Table1</code> is a base table. Generates a full base table called <code>Table1</code> or modifies existing full base table <code>Table1</code> .
<code>type = "base"</code> <code>base-table = "Table2"</code>	<code>table="Table1"</code>	DataDeploy assumes <code>Table1</code> is a delta table. Effectively merges rows from delta table <code>Table1</code> into base table <code>Table2</code> .
<code>type = "delta"</code> <code>base-table = "Table2"</code>	<code>table="Table1"</code>	DataDeploy assumes <code>Table1</code> is a delta table based on the full base table <code>Table2</code> . Generates a delta table called <code>Table1</code> or modifies existing delta table <code>Table1</code> . Does not update <code>Table2</code> with delta or any other type of data.

- 14. Columns to update:** In the `update` section, you must also select at least one column to update from the row(s) you specified earlier in the `select` section. You select columns by naming matching criteria in `column` tag attributes just as you did in the `select` section. All of the attributes shown in the table in Item 13, "Update type and related data," are supported in the `column` tag as well.



- 15. **SQL section:** The optional `sql` section lets you create SQL commands that override system defaults and execute automatically during deployment. The `sql` element supports three attributes: `action`, `user-action`, and `type`. Details are as follows:

Attribute	Value	Description
action	create	Lets you define your own SQL <code>CREATE TABLE</code> command for table creation during deployment. Commands set by this attribute override the default DataDeploy schema for creating tables. The default schema is <code>SELECT * FROM TABLENAME</code>
	show	Lets you define your own SQL <code>SELECT</code> command for the <code>show-table</code> operation. Commands set by this attribute override the default DataDeploy schema.
	exist	Lets you define database-specific queries to check for the existence of a table. Commands set by this attribute override the default DataDeploy schema.
	tracker-exist	Lets you define database-specific queries to check for the existence of the tracker table. Commands set by this attribute override the default DataDeploy schema.
	tracker-create	Lets you define your own SQL <code>CREATE TABLE</code> command for tracker table creation during deployment. Commands set by this attribute override the default DataDeploy schema.
user-action	<i>anyname</i>	<p>Lets you define any arbitrary SQL command(s) for execution during deployment. For example:</p> <pre><sql user-action="showview" type=query> Arbitrary SQL commands... </sql></pre> <p>The commands specified here execute only if you set the <code>iwdd-op=do-sql</code> and <code>user-op=anyname</code> options on the command line when you invoke DataDeploy. Because the <code>action</code> and <code>user-action</code> attributes are controlled by mutually exclusive command line options, you cannot execute both attributes at the same time (i.e., within the same deployment).</p>

type	query	If user-action is set, you must also set type. Setting type="query" specifies that user-action will be a query.
	update	If user-action is set, you must also set type. Setting type="update" specifies that user-action will be an update.

Note that it is not necessary for the statements in the <select> and <update> elements to match the table schema in an <sql> element.

16. **Server section:** The server section lets you specify a set of server-specific parameters. A deployment that is expected to run on a server in a three-tier system must have exactly one server section. The bind tag lets you specify where on the server machine the DataDeploy server will listen. Each server section must have exactly one bind section. In a bind section, the port attribute is always required, while the ip attribute is required only if the server machine has more than one available IP address. The optional allowed-hosts element lets you specify which hosts are allowed to connect to the DataDeploy server. If you include an allowed-hosts element, its host subelement must have an addr value in the form of an alphanumeric machine name or an IP address. The optional for-deployment element lets you define several client attributes just as you did in the database section (see Item 11). These attributes are: db, table, user, password, and timeout. If you set these attributes here, they override any settings for the same attributes in the client-side database section. An alternative to including a server section in a client/server configuration file is to have a separate file containing just a server section. This arrangement allows you to separate client and server information into different files, which can reside on different machines.

Sample TeamXpress-to-XML Configuration File

The following file configures a typical deployment from TeamXpress to an XML file. The `xml-formatted-data` tag has a single attribute, `file`, which specifies the absolute path and file name of the destination file. A `destinations` section can have any number of `xml-formatted-data` elements, or a combination of `xml-formatted-data` and `database` elements. When deploying to an XML file, you can also remap field column tags as shown on page 198.

```
<deployment name="TeamXpress-to-xml">
  <source>
    <!-- Pull data tuples from TeamXpress EA's -->
    <TeamSite-extended-attributes
      options="full"
      area="/default/main/dev/STAGING" >
      <path name="." />
    </TeamSite-extended-attributes>
  </source>
  <destinations>
    <xml-formatted-data file="/u/temp/someTable.xml" />
  </destinations>
</deployment>
```

The following sample file shows the default format of a typical XML destination file:

```
<?xml version="1.0"?>
<xml-tuple-data version="2.0">
  <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">small</tuple-field>
    <tuple-field name="key">size</tuple-field>
  </data-tuple>
  <data-tuple>
    <tuple-field name="path">mydir/f9</tuple-field>
    <tuple-field name="state">Original</tuple-field>
    <tuple-field name="value">blue</tuple-field>
    <tuple-field name="key">color</tuple-field>
  </data-tuple>
</xml-tuple-data>
```

Sample Database-to-Database Configuration File

```

<deployment name="db-to-db">
  <source>
    <!-- Pull data tuples from database -->
    <database db="server"
      user="DBA"
      password="SQL"
      table="staging">
      <fields>
        <field name="path" column="Path" />
        <field name="key" column="KeyName" />
        <field name="value" column="Value" />
        <field name="state" column="State" />
      </fields>
    </database>
  </source>
  <destinations>
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="someTable">
      <select>
        <column name="Path"
          value-from-field="path" />
        <column name="KeyName"
          value-from-field="key" />
      </select>
      <update>
        <!-- Update column 'Value' to contain the -->
        <!-- current EA value, and update column 'State' -->
        <!-- to contain the current state. -->
        <!-- This is a k-v-p specification -->
        <column name="Value"
          value-from-field="value" />
        <column name="State"
          value-from-field="state" />
      </update>
    </database>
  </destinations>
</deployment>

```

In this file, the `field` elements specify which columns in the source database DataDeploy will use when building a tuple for each row. The `select` element chooses rows to update in the destination database. It will choose rows only having unique combinations of the values named in the column subelements (in this case, `path` and `key`). See “Sample TeamXpress-to-XML Configuration File” on page 196 for an example of XML destination file format.

Sample Database-to-XML Configuration File

The following file configures a deployment from a database to an XML file, including remapped field column tags (as opposed to the default output shown on page 196):

```
<deployment name="db-to-xml">
  <source>
    <!-- Pull data tuples from database -->
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="tupleTable">
      <fields>
        <field name="path" column="EPath" />
        <field name="key" column="EKeyName" />
        <field name="value" column="EValue" />
        <field name="state" column="EState" />
      </fields>
    </database>
  </source>
  <destinations>
    <xml-formatted-data file="/tmp/tupleTable.xml">
    </xml-formatted-data>
  </destinations>
</deployment>
```


The resulting XML output file is as follows:

```
<?xml version="1.0"?>
<xml-tuple-data version="2.0">
  <data-tuple>
    <tuple-field name="NEWpath">mydir/f9</tuple-field>
    <tuple-field name="NEWstate">Original</tuple-field>
    <tuple-field name="NEWvalue">small</tuple-field>
    <tuple-field name="NEWkey">size</tuple-field>
  </data-tuple>
  <data-tuple>
    <tuple-field name="NEWpath">mydir/f9</tuple-field>
    <tuple-field name="NEWstate">Original</tuple-field>
    <tuple-field name="NEWvalue">blue</tuple-field>
    <tuple-field name="NEWkey">color</tuple-field>
  </data-tuple>
</xml-tuple-data>
```



Sample XML-to-Database Configuration File

The following file configures a typical deployment from an XML file to a database:

```
<deployment name="xml-to-db">
  <source>
    <!-- Pull data tuples from XML file -->
    <xml-formatted-data file="/u/iw/wcuan/billTable.xml" >
      <fields>
        <field name="path" element="path" />
        <field name="key" element="key" />
        <field name="value" element="value" />
        <field name="state" element="state" />
      </fields>
    </xml-formatted-data>
  </source>
  <destinations>
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="TableFromXML">
      <select>
        <column name="Path"
          value-from-field="path" />
        <column name="KeyName"
          value-from-field="key" />
      </select>
      <update>
        <!-- Update column 'RelatedValue' to contain the -->
        <!-- current EA value, and update column 'status' -->
        <!-- to contain the current state. -->
        <!-- This is a k-v-p specification -->
        <column name="Value"
          value-from-field="value" />
        <column name="State"
          value-from-field="state" />
      </update>
    </database>
  </destinations>
</deployment>
```

In this file, the `field` elements specify which attributes in the source XML file DataDeploy will use when building a tuple for each Path-Key-Value-State item in the file. The `element` attribute can name any valid element; it is not limited to naming just the `path`, `key`, `value`, or `state` elements shown here.



Sample XML-to-XML Configuration File

The following file configures a typical deployment from an XML file to another XML file. This is different than just copying the source file because it includes an in-flow substitution as described in the file comments. You can also include filters when configuring an XML-to-XML deployment, although that feature is not shown here.

```
<deployment name="xml-to-xml">
  <source>
    <!-- Pull data tuples from XML file -->
    <xml-formatted-data file="/u/iw/wcuan/billTable.xml" >
      <fields>
        <field name="path" element="path" />
        <field name="key" element="key" />
        <field name="value" element="value" />
        <field name="state" element="state" />
      </fields>
    </xml-formatted-data>
  </source>
  <substitution>
    <!-- Modify each tuple according to the following -->
    <!-- match/replace pairs. In this case: any path -->
    <!-- that contains the string 'WORKAREA/.../' will -->
    <!-- have the string replaced by 'STAGING/'; any -->
    <!-- path that contains 'EDITION/abcd' will be -->
    <!-- replace with '/This/Special/Path', and any -->
    <!-- tuple whose key starts with 'BEFORE' will be -->
    <!-- changed to begin with 'AFTER'. -->
    <field name="path"
      match="(.*)/WORKAREA/[^/]+/(.*)"
      replace="\1/STAGING/\2" />
    <field name="path"
      match="EDITION/abcd"
      replace="/This/Special/Path" />
    <field name="key"
      match="^BEFORE(.*)"
      replace="AFTER\1" />
  </substitution>
  <destinations>
    <xml-formatted-data file="/u/temp/someTable.xml" />
  </destinations>
</deployment>
```

In this file, the `field` elements specify which attributes in the source XML file DataDeploy will use when building a tuple for each Path-Key-Value-State item in the file.



Starting-State Base Table Configuration File

The following file generates the initial base table BT1 shown in the Starting State diagram on page 170:

```
<deployment name="staging">
  <source>
    <!-- Pull data tuples from TeamXpress EA's -->
    <TeamSite-extended-attributes
      options="full"
      area="/default/main/dev/STAGING" >
      <path name="." />
    </TeamSite-extended-attributes>
  </source>
  <destinations>
    <!-- Oracle8 on Unix -->
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="staging">
      <select>
        <column name="Path"
          value-from-field="path" />
        <column name="KeyName"
          value-from-field="key" />
      </select>
      <update type="base"
        state-field="state">
        <!-- Update column 'Value' to contain the -->
        <!-- current EA value, and update column 'State' -->
        <!-- to contain the current state. -->
        <!-- This is a k-v-p specification -->
        <column name="Value"
          value-from-field="value" />
        <column name="State"
          value-from-field="state" />
      </update>
    </database>
  </destinations>
</deployment>
```

Event 1 Configuration File

The following file configures the delta deployment shown in the Event 1 diagram on page 170:

```
<deployment name="delta">
  <source>
    <TeamSite-extended-attributes
      options="differential"
      base-area="/default/main/dev/STAGING"
      area="/default/main/dev/WORKAREA/$workarea" >
      <path name="." />
    </TeamSite-extended-attributes>
  </source>
  <destinations>
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="Delta_$$workarea">
      <select>
        <column name="Path"
          value-from-field="path" />
        <column name="Key"
          value-from-field="key" />
      </select>
      <update type="delta"
        base-table="staging"
        state-field="state">
        <column name="Value"
          value-from-field="value" />
        <column name="State"
          value-from-field="state" />
      </update>
    </database>
  </destinations>
</deployment>
```

Note that this file uses the parameter substitution `$workarea` in the `<database>` section. See “Parameter Substitutions” on page 175 for more information.



Event 2 Configuration File

The following file configures the delta deployment shown in the Event 2 diagram on page 170:

```
<deployment name="submit">
  <source>
    <TeamSite-extended-attributes
      options="differential"
      base-area="/default/main/dev/STAGING"
      area="/default/main/dev/WORKAREA/$workarea">
      <path filelist="/tmp/somefiles" />
    </TeamSite-extended-attributes>
  </source>
  <destinations>
    <database db="diver:1521:testdb"
      user="scott"
      password="tiger"
      table="Delta_$workarea">
      <select>
        <column name="Path"
          value-from-field="path" />
        <column name="Key"
          value-from-field="key" />
      </select>
      <update type="base"
        base-table="staging"
        state-field="state">
        <column name="Value"
          value-from-field="value" />
        <column name="State"
          value-from-field="state" />
      </update>
    </database>
  </destinations>
</deployment>
```


Chapter 10

Invoking DataDeploy

This chapter describes how to invoke DataDeploy from the command line, and the conditions under which the DataDeploy daemon runs as a service. You can also use the syntax shown here to invoke DataDeploy through an `iwat` trigger script or an external workflow task as described on page 151.

iwdd.ipl Command

Use the `iwdd.ipl` command to invoke DataDeploy from the command line, in an `iwat` trigger script, or as a workflow task. Usage is as follows. Note that `iwdd.ipl` resides in `dd-home/bin`.

Usage

```
iwdd.ipl cfg=configfile [deployment=deploymentname][iwdd-op=tableopname]
```

```
iwdd.ipl cfg=configfile [deployment=deploymentname][iwdd-op=do-sql] user-  
op=anyname mytable=anytable
```

```
iwdd.ipl remote-host=hostname [remote-port=portnumber][iwdd-  
op=serveropname]
```

Syntax

`iwdd.ipl` *cfg*

Invokes DataDeploy, and optionally performs table operations.

configfile

The name of the DataDeploy configuration file, including path name (either absolute or relative to the current directory).

deployment=*deploymentname*

Invokes DataDeploy as a client. Without this option, DataDeploy is invoked as a server.

deploymentname

The value of the `name` attribute of the `deployment` element.

iwdd-op=tableopname

Performs the table operation specified by *tableopname*. This is not a standalone option; you can only use it together with `deployment=deploymentname`.

tableopname

Displays or deletes tables as follows:

`show-table`: Displays an ASCII version of the table named by `table=` in the configuration file's `database` section for the specified deployment.

`drop-table`: Deletes the same table from the database.

`show-tracker`: Displays an ASCII version of the tracker table.

`drop-tracker`: Deletes the tracker table from the database.

iwdd-op=do-sql

Performs an SQL operation on the named table.

user-op=anyname

Performs the user-defined SQL operation defined by `user-action=anyname` in the DataDeploy configuration file's `sql` element. See Item 14 in "Sample File Notes" on page 181 section for more information. You must also set `mytable=anytable` whenever you set `user-op=anyname`.

iwdd remote-host

Performs server operations on the server specified in *hostname*.

hostname

The IP address or name of the server host.

remote-port=portnumber

Specifies the port number on the host. Defaults to 1949 if `remote-port` is not set.

iwdd-op=serveropname

Performs the server operation specified by *serveropname*. Defaults to `ping-server` if not set.

serveropname

ping-server: Returns a standard string to verify the server connection.

stop-server: Waits for current deployment to complete and then stops the server. All communication with the server is cut off after you issue this command.

kill-server: Stops the server immediately even if a deployment is running.

Examples

To invoke DataDeploy as a server based on the configuration file `/bin/conf/ddconfig.xml`:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml
```

To invoke DataDeploy as a client based on the configuration file `/bin/conf/ddconfig.xml` and the deployment named `ea-to-db`:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml deployment=ea-to-db
```

To delete the tracker table from the database:

```
iwdd.ipl cfg=/bin/conf/ddconfig.xml deployment=ea-to-db iwdd-op=drop-tracker
```

To stop the server on port 1234 of the host `examplehost`:

```
iwdd.ipl remote-host=examplehost remote-port=1234 iwdd-op=stop-server
```

To ping the server on port 1949 of the host `examplehost`:

```
iwdd.ipl remote-host=examplehost
```

Execute the following to invoke DataDeploy as a client to perform the SQL operation `showpaths` on the table `prtable`. In this example:

- The DataDeploy configuration file is `../conf/templating/extranet/pr.cfg`.
- `showpaths` is the value for the `user-op` attribute in the configuration file's `<sql>` element.
- `mytable="prtable"` is a parameter substitution for all occurrences of `$mytable` in the configuration file (see “Parameter Substitutions” on page 175 for more information).

```
iwdd.ipl cfg=../conf/templating/extranet/pr.cfg deployment="dosql" iwdd-  
op=do-sql user-op="showpaths" mytable="prtable"
```

Running DataDeploy as a Service

The Interwoven DataDeploy service automatically starts the DataDeploy daemon for DAS operation if the `iwsyncdb.cfg` file exists in `dd-home/conf`. If `iwsyncdb.cfg` does not exist, the Interwoven DataDeploy service starts the DataDeploy daemon for 3-tier operation.

Synchronizing OpenDeploy and Data Deploy

This chapter describes the configuration tasks you must perform to synchronize OpenDeploy with DataDeploy, and how to invoke a deployment after synchronization is complete.

Overview

You can configure your system to deploy file system assets and database assets in the same transactional deployment. This type of deployment is referred to as *synchronized deployment* throughout this chapter.

Deploying Different Types of Assets

File system assets are files (HTML, ASCII, etc.) that are typically deployed by previous releases of OpenDeploy. Database assets are TeamXpress extended attributes and data content records (DCRs) created through TeamXpress Templating. Deployment of database assets was not supported by previous releases of OpenDeploy.

A typical scenario for using synchronized deployment involves files that were generated via TeamXpress Templating. For example, after you configure synchronized deployment, if you deploy HTML files that were generated by rendering DCRs through presentation templates, the DCRs are also deployed to database tables residing on the production server. These actions occur as the result of a single synchronized deployment.

Synchronized deployment is intended primarily for deploying TeamXpress editions. The first development-to-production deployment of an edition will deploy all files and all database assets. Subsequent deployments will be dir-diffs between a subsequent edition on the production server and tuple differences between the current edition and the previous edition.

Synchronized deployment generates or updates one or more base tables on the production server. It does not generate delta tables. See “Client Configuration File” on page 217 for more information about what information is deployed to these base tables.

Note: Adding new files or database content to the production server by any method other than the edition deployment process will create data inconsistencies between assets on the development and production servers.

Configuration Task Categories

Tasks that you must perform to configure your system for synchronized deployment fall into two main categories:

- Ensuring that OpenDeploy configuration files are set up correctly on the development and production servers.
- Ensuring that DataDeploy configuration files are set up correctly on the development server and production servers.

The rest of this chapter describes:

- The software required for synchronized deployment.
- The files provided with OpenDeploy and DataDeploy to support synchronized deployment.
- An overview of what happens during synchronized deployment.
- OpenDeploy and DataDeploy configuration tasks that you must perform to set up synchronized deployment.
- How to invoke synchronized deployment.

Software Requirements

Synchronized deployment requires the following software:

- TeamXpress 1.1 on the development server.
- TeamXpress Templating 1.1 on the development server.
- TeamXpress OpenDeploy 4.5.1 on the development and production servers.
- TeamXpress DataDeploy 4.5.1 on the development and production servers.

Program and Configuration Files




The following files control synchronized deployment. All files are installed by default in `od-home/examples/ddsync`. You must manually move them to the locations shown below. See the sections following the table for configuration instructions and illustrations showing how these files interact.

File	Location	Description
<code>ddsync.ipl</code>	<code>dd-home/bin</code> on the development and production servers.	The DNR script invoked by OpenDeploy to execute DataDeploy. Do not edit this file.
<code>database.xml</code>	<code>dd-home/conf</code> on the production server.	The include file for the <code><database></code> element in <code>loaddb.cfg</code> . You must configure this file for your site. See “Configuring DataDeploy” on page 222 for more information.
<code>subxmlldb.template</code>	<code>dd-home/conf</code> on the development server.	The Template DataDeploy configuration file used by <code>iwsyncdb.ipl</code> as a basis for creating the <code>loaddb.cfg</code> DataDeploy configuration file. Do not edit this file.
<code>loaddb.cfg</code>	Generated on the production server; must be moved to <code>dd-home/conf</code> on the production server.	The generated DataDeploy configuration file used to run DataDeploy to update a database from generated XML files. Do not edit this file. You can, however regenerate it from the command line. See “Configuring DataDeploy” on page 222 for more information.

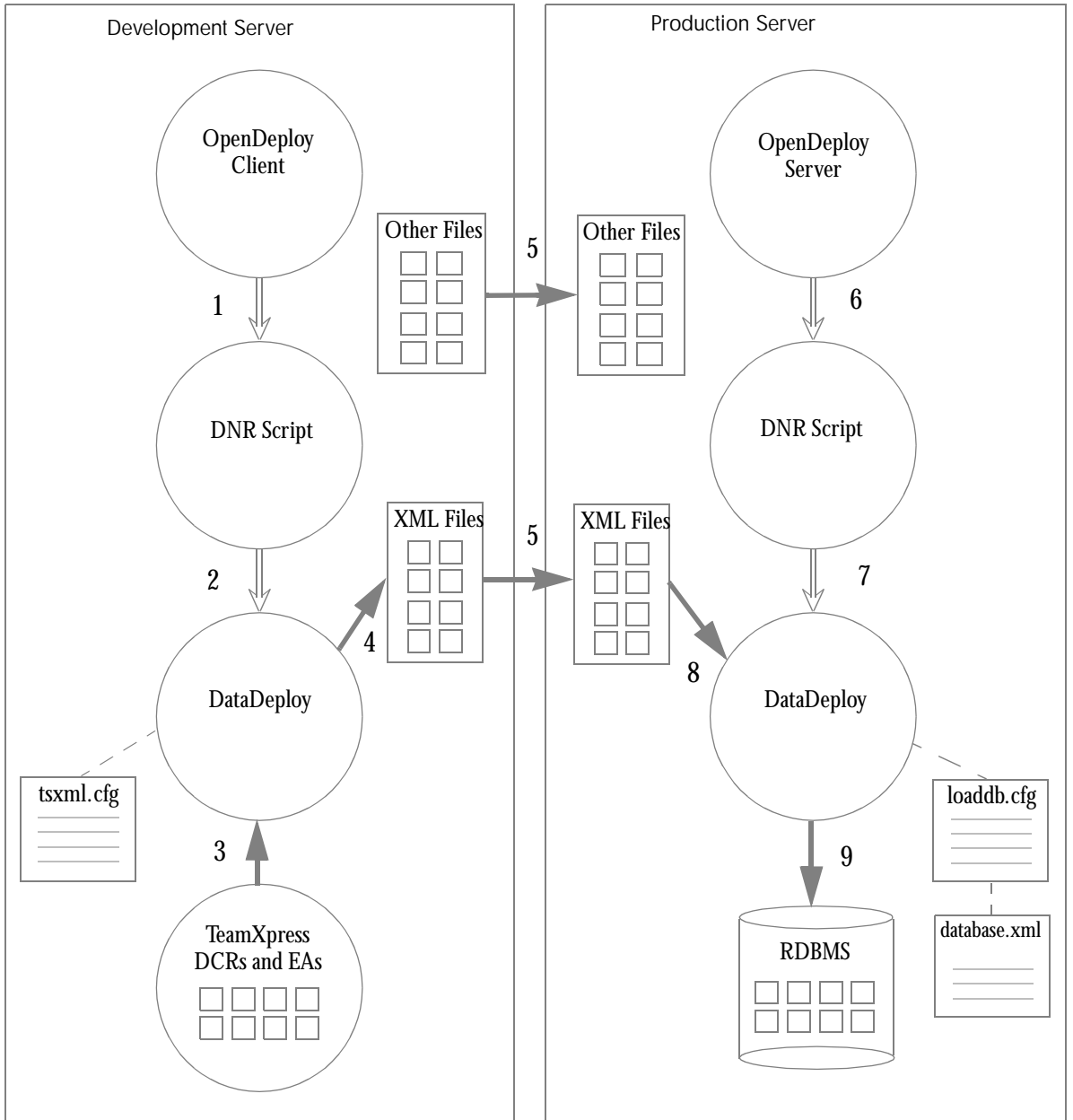
File	Location	Description
<code>tsxml.cfg</code>	<i>dd-home/conf</i> on the development server.	The DataDeploy configuration file used to run DataDeploy to generate XML files from TeamXpress extended attributes and DCRs. You must configure this file for your site. See “Configuring DataDeploy” on page 222 for more information.
<code>oddd_receive.cfg</code>	<i>od-home/conf</i> on the production server.	OpenDeploy server-side (production server) configuration file. You must configure this file for your site. See “Server Configuration File” on page 221 for more information.
<code>oddd_send.cfg</code>	<i>od-home/conf</i> on the development server.	OpenDeploy client-side (development server) configuration file. You must configure this file for your site. See “Configuring OpenDeploy” on page 217 for more information.

Synchronized Deployment Process

The following diagram shows the deployment of file system assets and database assets from a development server to a production server. There are two main deployment paths: one for the file system assets (labeled “Other Files” in the diagram) and one for the database assets such as TeamXpress DCRs and extended attributes. The diagram uses the following symbols:

-  Flow of data such as DCRs and extended attributes.
-  Flow of OpenDeploy and DataDeploy processes.
-  Flow of OpenDeploy and DataDeploy processes.

The section following the diagram explains all deployment steps in detail.



Synchronized Deployment

Diagram Key

This section explains the actions shown in the preceding diagram. For these actions to take place, you must have already configured the OpenDeploy and DataDeploy files as described in this chapter. You must also have already started the OpenDeploy daemon on the production server by executing the `iwdeploy -s` command described in “” on page 222.

1. On the development server, a user invokes OpenDeploy in transactional mode from the command line via the `iwdeploy -T` command. See “Invoking Synchronized Deployment” on page 227 for syntax details. The client- and server-side OpenDeploy configuration files for this deployment must be configured to deploy file system assets per a normal deployment, and must also contain deploy-and-run (DNR) scripts for database asset deployment. See “Configuring OpenDeploy” on page 217 for details about setting up these OpenDeploy configuration files.

The OpenDeploy client starts the client-side DNR script.

2. The client-side DNR script invokes DataDeploy as configured by `tsxml.cfg`. See “” on page 222 for details about `tsxml.cfg`.
3. DataDeploy reads the TeamXpress database assets (DCRs and extended attributes) residing on the development server.
4. DataDeploy performs a TeamXpress-to-XML deployment, generating XML files based on the TeamXpress database assets. See “Sample TeamXpress-to-XML Configuration File” on page 196 for more information about this type of deployment.
5. OpenDeploy deploys the generated XML files and the original file system assets to the production server.
6. OpenDeploy is already running on the production server (it was started as a daemon via `iwdeploy -s` prior to Step 1 above). The OpenDeploy daemon starts the server-side DNR script.
7. The server-side DNR script invokes DataDeploy as configured by `loaddb.cfg` and `database.xml`.

8. DataDeploy reads the generated XML files that were deployed from the development server.
9. DataDeploy performs an XML-to-database deployment, populating the database on the production server with tuples from the generated XML files. See “Sample XML-to-Database Configuration File” on page 200 for more information about this type of deployment.

Configuring OpenDeploy

This section describes the steps you must perform to configure OpenDeploy for synchronized deployment at your site. Configuration steps are:

1. Edit the client configuration file `oddd_send.cfg` to control OpenDeploy client execution on the development server.
2. Edit the server configuration file `oddd_receive.cfg` to control OpenDeploy server execution on the production server.
3. Start the OpenDeploy daemon on the production server.

The following sections describe these steps in detail.

Client Configuration File

The sample client configuration file `oddd_send.cfg` contains general configuration information and four DNR scripts (all based on `ddsync.ipl`). Each DNR script invokes DataDeploy differently depending upon whether the deployment is *full* or *differential*, and whether DataDeploy is invoked on the development or production server. Full and differential deployments in the context of synchronized deployment are defined as follows:

- With full deployment, new base tables are created for each TeamXpress Templating data type named in `tsxml.cfg`. Full deployment is typically done once, as the first synchronized deployment on your system. If you execute a full deployment more than once, existing base tables are overwritten with new base tables upon each execution.
- With differential deployment, existing base tables are updated with any data that is new or changed since the last deployment.

To configure `oddd_send.cfg` for your site, you must edit the general configuration information and all four DNR scripts as follows.

Note: If a file named `oddd_send.cfg` already exists on your system, the sample configuration file should be integrated into the existing version.

1. Open `od-home/conf/oddd_send.cfg`. You will see two main sections labeled `Dir-Diff Deployment with Full Tuple Deploy` and `Dir-Diff Deployment with Differential Tuple Deploy`. Each section contains general configuration information and two DNR scripts.
2. Change all references to `/local/iw-home` to reflect the location of the actual OpenDeploy home directory.
3. The following line specifies the development server location of the edition that you intend to deploy in the initial full deployment:

```
area=/default/main/dev/EDITION/snapshot
```

In each occurrence of this line, change `/default/main/dev/EDITION/snapshot` to reflect the development server location of the edition that you will deploy. This edition is depicted as “TeamXpress DCRs and EAs” in the diagram on page 215. This line applies to file assets during a full deployment.

4. The following line specifies the production server location of the destination for deployed file system assets:

```
remote_directory=/tmp/Branch1
```

In each occurrence of this line, change `/tmp/Branch1` to reflect the production server location of the destination for deployed file system assets. These assets are depicted as “Other Files” in the diagram on page 215. This line applies to file assets during both full and differential deployments.

5. The following line specifies the development server location of the more recent edition that you will use for comparison during differential deployment (in which two editions are compared and only differences are deployed):

```
area=/default/main/dev/EDITION/snapshot2
```

In each occurrence of this line, change `/default/main/dev/EDITION/snapshot2` to reflect the development server location of the more recently created edition that you will use for comparison. In the example shown here, `snapshot2` is compared with `snapshot`, and the differences are then deployed.

- The following line specifies the production server location of the destination for generated XML files during the initial full deployment:

```
remote_directory=/tmp/production/dumpdir
```

In each occurrence of this line, change `/tmp/production/dumpdir` to reflect the production server location of the destination for generated XML files. These files are depicted as “XML Files” in the diagram on page 215. This line applies to XML files during a full deployment.

- The following line specifies the development server location of the generated XML files during the initial full deployment:

```
area=/tmp/development/dumpdir
```

In each occurrence of this line, change `/tmp/development/dumpdir` to reflect the development server location of the generated XML files. These files are depicted as “XML Files” in the diagram on page 215. This line applies to XML files during a full deployment.

- The following line specifies the production server location of the destination for generated XML files during a differential deployment:

```
remote_directory=/tmp/production/deltadumpdir
```

In each occurrence of this line, change `/tmp/production/deltadumpdir` to reflect the production server location of the destination for generated XML files during a differential deployment. These files are depicted as “XML Files” in the diagram on page 215.

- The following line specifies the development server location of the generated XML files during a differential deployment :

```
area=/tmp/development/deltadumpdir
```

In each occurrence of this line, change `/tmp/development/deltadumpdir` to reflect the development server location of the generated XML files during a differential deployment. These files are depicted as “XML Files” in the diagram on page 215.

10. Wherever `/temp/development`, `/temp/production`, `dumpdir`, and `deltadumpdir` occur in the DNR scripts, change them to match the values determined in the preceding steps.
11. Where `snapshot2` occurs in the differential deployment DNR script, change it to match the development server location of the more recently created edition that you will use for comparison in a differential deployment.

Syntax of `ddsync.ipl`

This section shows the full syntax for the `ddsync.ipl` DNR script.

Usage

```
ddsync.ipl area_top dump_dir dump full area
ddsync.ipl area_top dump_dir dump differential area basearea
ddsync.ipl area_top dump_dir load full
ddsync.ipl area_top dump_dir load differential
```

<code>area_top</code>	The absolute path to top of the area directory.
<code>dump_dir</code>	The relative path to the dump directory in <code>area</code> .
<code>dump</code>	Dumps TeamXpress metadata to generated XML (“dump”) files.
<code>load</code>	Populates database table(s) with data from generated XML (“dump”) files.
<code>full</code>	Specifies entire area traversal.
<code>differential</code>	Specifies comparison between two areas.
<code>basearea</code>	Specifies the originally-created area for use in comparison during a differential deployment. Valid only if <code>differential</code> is set.
<code>area</code>	Specifies the current area (i.e., the more recently created area for use in comparison during a differential deployment).

Logging ddsync.ipl Execution

A log of `ddsync.ipl` execution is maintained in `dd-log-home/ddsync_dump_load.log`.

Supported OpenDeploy Modes

Whenever you invoke a synchronized deployment, you must execute OpenDeploy in forward dir-diff mode. TeamXpress-based, file list, and reverse deployments are not supported. See “Invoking Synchronized Deployment” on page 227 for appropriate command-line syntax for invoking in dir-diff mode.

Configuration File Location

After you configure `oddd_send.cfg`, ensure that it resides in `od-home/conf`.

Server Configuration File

The sample server configuration file `od-home/conf/oddd_receive.cfg` is shown below. You must edit it to reflect your system’s port number and the TeamXpress server name. You must also edit the values of `/tmp/Branch1` and `/tmp/production` so that they match the values you entered in `oddd_send.cfg`.

```
#
# Test file for self-host deployment of some dir
#
#     SERVER-SIDE
#
port=1999
timeout=600
#require_abs_script_path=y
TeamSite_server=pegasus
    #key_file=/secrets/some-shared-secret-file
    allowed_directory = /tmp/Branch1
    allowed_directory = /tmp/production
;
```

Configuration File Location

After you configure `oddd_receive.cfg`, ensure that it resides in `od-home/conf`.

Starting the OpenDeploy Server Daemon

After configuring the server configuration file, start the OpenDeploy server daemon on the production server by executing the following command:

```
iwdeploy -s -f oddd_receive.cfg
```

Configuring DataDeploy

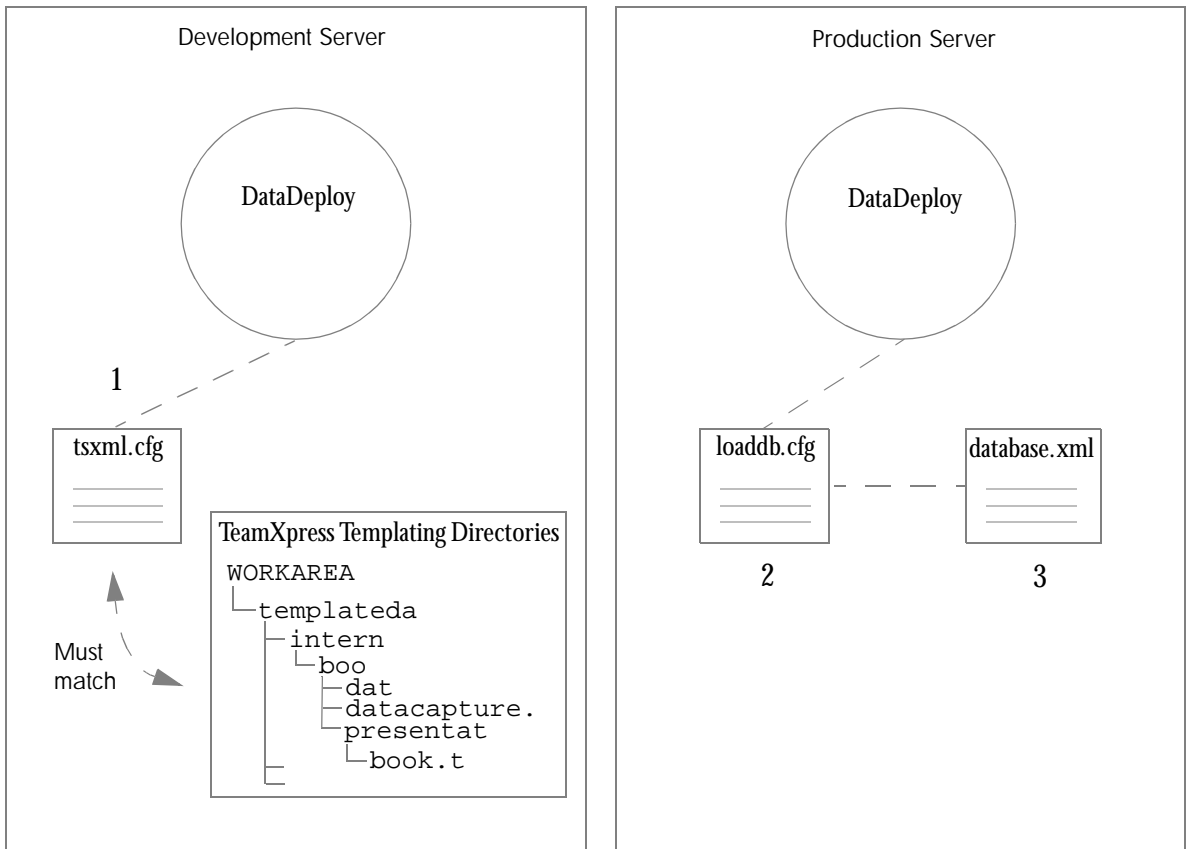
This section describes the steps you must perform to configure DataDeploy for synchronized deployment at your site. Configuration steps are:

1. Edit the configuration file `tsxml.cfg`.
2. Generate the configuration file `loaddb.cfg`.
3. Edit the configuration include file `database.xml`.
4. Verify that the configuration files reside in the appropriate directories.

The following sections describe these steps in detail.

Interaction Between Files

The following diagram shows the interaction between DataDeploy configuration files during a synchronized deployment. The section following the diagram explains each component in detail.



Synchronized Deployment: DataDeploy Configuration Files

Diagram Key

1. When DataDeploy is invoked on the development server via the `ddsync.ip1` DNR script, the deployment is based on the `dd-home/tsxml.cfg` file. Prior to invoking the synchronized deployment, you must have edited this file manually so that it names each TeamXpress Templatng data category and type that will be deployed to a generated XML file.

Data categories and types are determined by examining the `templatedata` directory structure that was set up as described in the TeamXpress Templatng documentation. By default, `tsxml.cfg` contains the example data categories and types that are distributed with TeamXpress



Templating. The example in the diagram shows the part of this directory structure containing the `book` data type within the `internet` data category. See “Editing `tsxml.cfg`” on page 224 for details about how to list data categories and types.

You do not need to list every data category and type from the `templatedata` directory structure in `tsxml.cfg`. You only need to list the data types that you intend to deploy. However, each data type that you list in `tsxml.cfg` must exist in the `templatedata` directory structure. If you list data types in `tsxml.cfg` that do not exist in the directory structure, the deployment will fail.

Each data type that you list in `tsxml.cfg` will be deployed to its own generated XML file.

2. When DataDeploy is invoked on the production server via the `ddsync.ip1` DNR script, the deployment is based on the `dd-home/loadddb.cfg` file. Prior to invoking the synchronized deployment, you must have generated this file on the development server and then moved it to `dd-home` on the production server. You should not edit `loadddb.cfg` directly; doing so will create inconsistencies between assets on the development and production servers. See “Generating `loadddb.cfg`” on page 226 for details.
3. The information in `loadddb.cfg` is supplemented by `database.xml`, which is analogous to an include file. The `database.xml` file is named in the `<data-deploy-elements>` element in `loadddb.cfg`. You must edit `database.xml` to configure it for your site. See “Editing `database.xml`” on page 227 for details. See “Configuration File Details and Examples” for information about `<data-deploy-elements>` syntax.

Editing `tsxml.cfg`

Prior to invoking a synchronized deployment, you must edit `tsxml.cfg` as described in this section and ensure that it is installed in `dd-home/conf` on the development server. The following excerpts are from the version of `tsxml.cfg` that is shipped with DataDeploy. See the diagram key following the diagram for details.

```

<data-deploy-configuration>
  <client>
    <!-- -->
    <!-- Parameters: -->
    <!-- mode = { full | differential } -->
    <!-- -->
    <!-- if mode == full -->
    <!-- mybasearea = dummy -->
    <!-- myarea = absolute vpath to any area -->
    <!-- -->
    <!-- if mode == differential -->
    <!-- mybasearea = absolute vpath to prev edition -->
    <!-- myarea = absolute vpath to curr edition -->
    <!-- -->

    <deployment name="TeamXpress_metadata">
      <source>
        <TeamSite-extended-attributes
          options = "wide,$mode"
          base-area = "$mybasearea"
          area = "$myarea" >
          <path name = "."
            visit-directory = "deep" />
        </TeamSite-extended-attributes>
      </source>
      <destinations>
        <xml-formatted-data file="TeamSite_metadata.dump" />
      </destinations>
    </deployment>

    <deployment name="internet_book">
      <source>
        <TeamSite-templating-records
          options = "wide,$mode"
          base-area = "$mybasearea"
          area = "$myarea" >
          <path name = "templatedata/internet/book"
            visit-directory = "deep" />
        </TeamSite-templating-records>
      </source>
      <destinations>
        <xml-formatted-data file="internet_book.dump" />
      </destinations>
    </deployment>
  </client>

```

Deployment section for extended attributes ¹

Deployment section for first data category/type ²

Sample File Notes

- 1. Deployment section for extended attributes:** Configures deployment of TeamXpress extended attributes to generated XML (“dump”) files. Do not edit this section of `tsxml.cfg`.
- 2. Deployment section for first data category/type:** Configures deployment of DCRs to generated XML (“dump”) files. The section shown in this example instructs DataDeploy to execute a deployment that creates a single XML file (`internet_book.dump`) containing DCRs for the `internet/book` data category/type. Note that the `internet` data category and `book` data type match the data category and type shown in the `templatedata` directory structure in the diagram on page 223. The default `tsxml.cfg` file included with DataDeploy contains several additional `<deployment>` sections for the other data categories and types that exist in the default TeamXpress Templating `templatedata` directory structure. Those sections are not shown here due to space constraints.

You must create a `<deployment>` section in `tsxml.cfg` for each data category and type that you intend to deploy via synchronized deployment. To do this, copy and edit the `<deployment>` section shown in this example, replacing all occurrences of `internet` and `book` with the appropriate data category and type (respectively) from your site’s `templatedata` directory structure. Repeat this process as necessary to create a `<deployment>` section in `tsxml.cfg` for each data type that you intend to deploy.

Generating `loaddb.cfg`

You must generate `loaddb.cfg` prior to invoking the first synchronized deployment. After that initial generation, it should not be necessary to regenerate `loaddb.cfg`. The `loaddb.cfg` file must reside on the production server. However, the TeamXpress Templating information needed to generate `loaddb.cfg` resides on the development server. Therefore, you must generate `loaddb.cfg` on the development server and then move it to `dd-home/conf` on the production server after it is generated.

Execute the following command on the development server to generate `loaddb.cfg` in the `target-path` directory of your choice. The path listed in `area-vpath` names the area containing the TeamXpress Templating directory structure as shown in “TeamXpress Templating Directories” in the diagram on page 223.

```
dd-home/bin/iwsyncdb.ipl -genloadcfg target-path/loaddb.cfg area-vpath
```

After `loaddb.cfg` is generated, move it to `dd-home/conf` on the production server. Then change the `<data-deploy-elements>` file path so that it contains the full pathname to the `database.xml` file.

Editing database.xml

The following sample `database.xml` file is distributed with DataDeploy. Prior to invoking a synchronized deployment, you must edit `database.xml` as described in this section and ensure that it is installed in `dd-home/conf` on the production server.

```
<data-deploy-elements>
  <database name      = "myproductiondb" ← Do not edit the name attribute.
    db                = "server"
    user              = "DBA" ← Edit the db, user, password,
    password          = "SQL"   and vendor attributes.
    vendor            = "SYBASE" />
</data-deploy-elements>
```

The database name `myproductiondb` is hardcoded in other program files and should not be edited in `database.xml`. However, you must edit the `db`, `user`, `password`, and `vendor` attributes per the syntax described in Item 11 on page 187 so that they are specific to your system.

Configuration File Locations

After configuring the DataDeploy configuration files, ensure that they reside in the locations shown in “Program and Configuration Files” on page 213.

Invoking Synchronized Deployment

After you have configured OpenDeploy and DataDeploy as described earlier in this chapter, you can invoke a synchronized deployment from the development server.

When you invoke synchronized deployment, you can chose either of the following deployment types:

- *Full deployment*, in which new base tables are created for each data type named in `tsxml.cfg`. Full deployment is typically done once, as the first synchronized deployment on your system. If you execute a full deployment more than once, existing base tables are overwritten with new base tables upon each execution.
- *Differential deployment*, in which existing base tables are updated with any data that is new or changed since the last deployment.

Issue the following command on the development server to execute a full deployment:

```
iwdeploy -T -f oddd_send.cfg full_tuple_deploy
```

Issue the following command on the development server to execute a differential deployment:

```
iwdeploy -T -f oddd_send.cfg delta_tuple_deploy
```

If any part of a synchronized deployment fails, the entire deployment is restored back to the previous state. For example, if the DataDeploy component fails, the files deployed by OpenDeploy files are reverted. If the OpenDeploy component fails, the DataDeploy component will not be invoked.

Section 3: OpenDeploy Administration

-
- Installing OpenDeploy
 - Syntax and Options
 - Configuration Files
 - Configuration File Options
 - Advanced Features
 - Deployment Scenarios

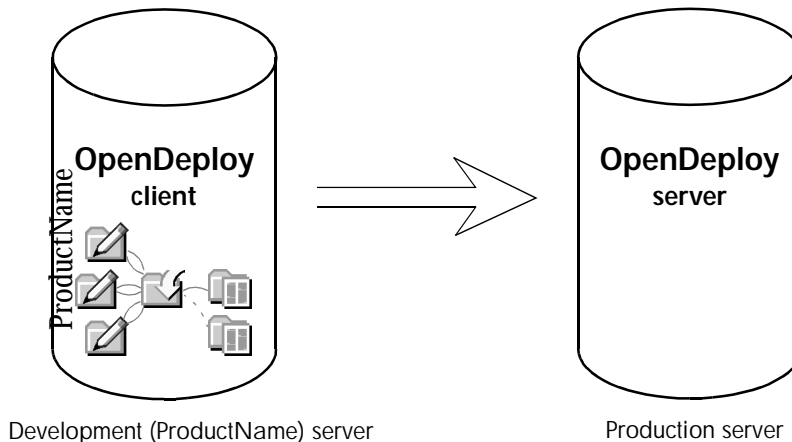


INTERWOVEN

Chapter 12

Installing OpenDeploy

OpenDeploy consists of a deployment client which resides on the TeamXpress or development server, and a deployment server which resides on the production server. Before using OpenDeploy, you must set up configuration files for the types of deployment you want (see Chapter 17, “Deployment Scenarios”).



The rest of this chapter describes how to install the OpenDeploy client and server. Installation on a UNIX system such as Solaris is covered in the following section. Installation on a Windows NT/2000 system follows the UNIX section.

UNIX

Before You Begin

You should perform the following tasks before installing the OpenDeploy client and server on a UNIX system:

1. If you are installing the OpenDeploy client on a system without TeamXpress, you will need to specify a directory where the OpenDeploy tar file will be loaded and uncompressed. It is recommended that you determine which directory this will be before starting the installation procedure.
2. During installation of the OpenDeploy server, you will be prompted to specify the production server's port number, the name of the production server, and the default directory on the production server where website content will be deployed. It is recommended that you determine this information before starting the installation procedure.

Installing the OpenDeploy Client

The following sections describe how to install the OpenDeploy client on development servers with and without TeamXpress.

Development Server With TeamXpress

To install the OpenDeploy client on a development server that has TeamXpress installed:

1. Copy the OpenDeploy tar file to your development server and decompress the tar file:

```
% gunzip -c opendeploy.tar.gz | (cd /`iwgethome`; tar xvpf -)
```
2. In the OpenDeploy home directory (`iw-home/opendeploy`), run `iwinstalloid`:

```
% cd iw-home/opendeploy
% install/iwinstalloid
```
3. When the OpenDeploy installation script prompts you to select the installation type, select 1 (source/client installation).
4. The OpenDeploy client will install on your development server. The installation process will create the log file `iw-home/log/deployEvents.log`.

Development Server Without TeamXpress

To install the OpenDeploy client on a development server that does not have TeamXpress installed:

1. Copy the OpenDeploy tar file into the directory on your development server where you want to uncompress and expand it, and decompress the tar file:

```
% gunzip -c opendeploy.tar.gz | (cd /parent_dir; tar xvpf -)
```

2. In the OpenDeploy home directory (*parent_dir/opendeploy*), run `iwinstallod`:

```
% cd parent_dir/opendeploy
```

```
% install/iwinstallod
```

3. When the OpenDeploy installation script prompts you to select the installation type, select 1 (source/client installation).
4. The OpenDeploy client will install on your development server. The installation process will create the log file `opendeploy/log/deployEvents.log`.

Installing the OpenDeploy Server

To install the OpenDeploy server:

1. Copy the OpenDeploy tar file into the directory on your production server where you want to uncompress and expand it. Decompress the tar file:

```
% gunzip -c opendeploy.tar.gz | (cd /parent_dir; tar xvpf -)
```

2. In the OpenDeploy home directory (*parent_dir/opendeploy*), run the `iwinstallod` program:

```
% cd parent_dir/opendeploy
```

```
% install/iwinstallod
```

3. When the OpenDeploy installation script prompts you to select the installation type, select 2 (target/server installation).
4. The OpenDeploy installation script will prompt you to specify some default parameters, such as port number, the name of the production server, and the directory to deploy to. These parameters will be used in a simple default configuration file, which the installation script will display. You can change any of these parameters by modifying the default configuration file or by using a different configuration file when you invoke the `iwdeploy` server.

The OpenDeploy server installation script installs the following files:

<code>/etc/iwopendeploy.cfg</code> (if TeamXpress is not installed)	Contains the directory path to the OpenDeploy home directory.
<code>/etc/init.d/iw.deploy</code>	The start/stop script for the OpenDeploy server.
<code>/etc/rc3.d/S80iw.deploy</code>	Hard link to <code>/etc/init.d/iw.deploy</code> (starts the OpenDeploy server at UNIX system startup time).
<code>/etc/rc3.d/K80iw.deploy</code>	Hard link to <code>/etc/init.d/iw.deploy</code> (stops the OpenDeploy server at UNIX system shutdown time).
<code>opendeploy/conf/iwodserver.cfg</code>	Default server configuration file.
<code>opendeploy/log</code>	Contains OpenDeploy log files.

OpenDeploy for UNIX creates a server log file with the level of verbose logging specified when the OpenDeploy server is invoked. This log is automatically generated in `opendeploy/iwdeploy.log`. If this log file starts to take up too much space, you can stop the OpenDeploy service, save the log file in another location, and restart the service. If the OpenDeploy server does not appear to be running, check this log file for information about the possible causes.

Uninstalling OpenDeploy

To uninstall OpenDeploy:

In the OpenDeploy home directory (`opendeploy`) on the development server, run the `iwuninstalled` program:

```
% cd iw-home/opendeploy
% install/iwuninstalled
```

Repeat the process on the production server.

Invoking Deployment

Before you invoke the OpenDeploy client and server, you must have configuration files set up for both the client and the server (see Chapter 14, “Configuration Files”). You should also have a good understanding of OpenDeploy syntax and options (see Chapter 13, “Syntax and Options”).

You can invoke deployment either manually or through `/etc/init.d/iw.deploy`.

To invoke deployment manually:

1. As superuser on the production server, invoke the `iwdeploy` server, either manually:¹

```
% iwdeploy -s -fd destConfigFile -v level -t tempFilePath &
```

or by editing the `/etc/init.d/iw.deploy` script to use the options you want, and invoking it:

```
% /etc/init.d/iw.deploy start
```

2. On the development server, invoke the `iwdeploy` client, either manually or through custom scripts using the TeamXpress suite of command triggers. The arguments you use for the `iwdeploy` client will depend on the type of deployment you want to invoke.

Windows NT/2000

Installing the OpenDeploy Client and Server

To install the OpenDeploy client on the development server:

1. Double-click on the self-extracting installation file `OpenDep1.exe`.
2. The installation files will extract themselves and begin installing OpenDeploy. Follow the directions in the onscreen installation prompts.
3. After completing the installation tasks described in the prompts, you must also make sure that the `TMP` environment variable is set on the production server. Select **Settings > Control Panel** from the Windows NT/2000 **Start** menu.

1. If you have just killed the `iwdeploy` server process and are now invoking it again, it can fail to bind to the port. Wait three minutes for the TCP/IP connection, then try again.

4. Open the **System** Control Panel.
5. Select the **Environment** tab. Scroll through the list of variables in the System Variables window.
If `TMP` appears in that window, then the variable is set and the OpenDeploy installation procedure is finished. You do not need to perform steps 6-10.
Note: Make sure that the `TMP` variable you see is the System environment variable, not the User environment variable.
If the `TMP` variable does not appear, then continue to steps 6-10 to set the variable.
6. To set the `TMP` variable, click on one of the settings in the System Variables window.
7. Change the setting in the **Variable** box to `TMP` (this change will not alter the existing system variable; it will add a new one).
8. Change the setting in the **Value** box to the location where you want to put temporary files (usually `C:\TEMP`).
9. Click **OK**.
10. You will need to reboot the server before using OpenDeploy.

The installation procedure sets a Registry key

(`HKEY_LOCAL_MACHINE\Software\Interwoven\OpenDeploy`) with three values:

<code>od-home</code>	The directory where OpenDeploy was installed (default: <code>C:\Program Files\Interwoven\OpenDeploy</code>)
<code>od-loghome</code>	The directory for OpenDeploy's log files (default: <code>C:\Program Files\Interwoven\OpenDeploy</code>)
<code>Version</code>	Current version of OpenDeploy

To install the OpenDeploy server:

Repeat steps 1-10 (above) on the production server.

Uninstalling OpenDeploy

To uninstall OpenDeploy:

1. Select **Start > Settings > Control Panels**.
2. Double-click on the **Add/Remove Programs** Control Panel icon.
3. Locate **Interwoven OpenDeploy** in the Control Panel, and click **Add/Remove**. UninstallShield will detect the existing OpenDeploy installation and remove the software.

Invoking OpenDeploy

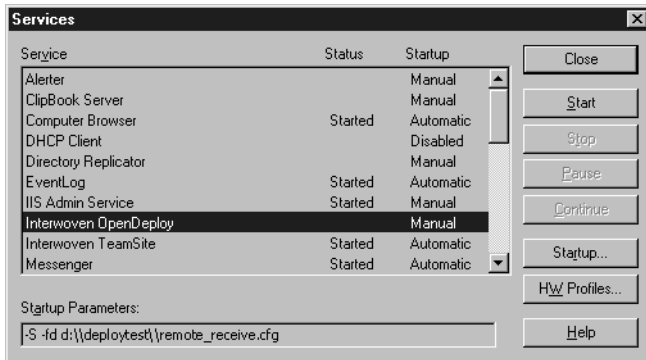
Before you invoke the OpenDeploy client and server, you must have configuration files set up for both the client and the server (see Chapter 14, “Configuration Files”). You should also have a good understanding of OpenDeploy syntax and options (see Chapter 13, “Syntax and Options”).

Invoking the OpenDeploy Server

To invoke the OpenDeploy server:

1. On the production server, select **Settings > Control Panel** from the Windows NT/2000 **Start** menu.
2. Open the **Services** Control Panel.
3. Select OpenDeploy from the list of services.
4. Type all the arguments you want to use (e. g. configuration file and type of logging) in the **Startup Parameters** box. When specifying paths to files, be sure to use double backslashes (\\) anywhere you would normally use single backslashes (\).

If OpenDeploy is invoked as a service, it will use the `-s` (server) option by default. If no configuration file is specified on the command line, OpenDeploy will use the default configuration file (`OpenDeploy/conf/iwodserver.cfg`).



The Windows NT Services Window

5. Click the **Start** button.

OpenDeploy for Windows NT/2000 creates a server trace file with the level of verbose logging specified when the OpenDeploy server is invoked. This log is automatically generated in the location you specify at the time of installation, with the name `deploySvrTracedate.id.txt`. Every restart of the service will create a new log file. If this log file gets too large, you can stop the OpenDeploy service, save the log file in another location, and restart the service.

To check the status of the OpenDeploy server, use the Windows NT/2000 Task Manager. If the OpenDeploy service does not appear, check the trace file.

Invoking the OpenDeploy Client

Before you can invoke the OpenDeploy client, you must invoke the OpenDeploy server.

To invoke the OpenDeploy client:

You can invoke the OpenDeploy client (on the development server) manually or through custom scripts. The OpenDeploy client is manually invoked from the command prompt. The arguments you use for the `iwdeploy` client will depend on the type of deployment you want to invoke.

Chapter 13

Syntax and Options

This chapter discusses the syntax and options of the `iwdeploy` command line tool (CLT).

iwdeploy Syntax

The syntax for `iwdeploy` is exactly the same in both UNIX and Windows NT/2000—only the means of invoking it differs. If you are invoking the `iwdeploy` server through the Windows NT/2000 Services Control Panel, you only need to type the arguments you want to use in the **Startup Parameters** box.

Server Usage

```
iwdeploy -S [-h] [-v] [{-f|-fd} destConfigFile] [-fs srcConfigFile]  
[-V level] [-t tempFilePath] [-i package_name] [-auth AuthFile]
```

Client Usage

```
iwdeploy [-h] [-v] [{-f|-fs} srcConfigFile] [-fd destConfigFile] [-r] [-T]  
[-V level] [-events] [-log option] [-logpath dirPath] [-t tempFilePath]  
[-o package_name] deployment_name [param=value]+
```

General options

<code>-h</code>	Displays usage message.
<code>-v</code>	Displays version.
<code>-V <i>level</i></code>	Specifies verbose logging level (1-4): the default is maximum verbosity (see page 243).



Server mode options

<code>-S</code>	Server mode.
<code>-fs srcConfigFile</code>	Specifies a source configuration file (only needed for reverse deployment, see page 337).
<code>{-f -fd} destConfigFile</code>	Specifies a destination configuration file. If you are using the default configuration file, you do not need to specify it. <code>-f</code> is an option provided for backward compatibility.
<code>-t tempFilePath</code>	Specifies a path for temporary file created during deployment (needs space for up to ~5 Mb).
<code>-i package_name</code>	Unpacks the deployment package created using the <code>-o</code> option.
<code>-auth AuthFile</code>	Specifies the authorization file to use (see page 258).

Client mode options

<code>{-f -fs} srcConfigFile</code>	Specifies a source deployment configuration file. If you are using the default configuration file, you do not need to specify this option. <code>-f</code> is an option provided for backward compatibility.
<code>-fd destConfigFile</code>	Specifies a destination configuration file (only needed for reverse deployment—see page 337).
<code>-r</code>	Pull mode (reverse deployment—see page 337).
<code>-T</code>	Transaction-based deployment (see page 242).
<code>-events</code>	Specifies logging for Event Reporting (see page 249).
<code>-log option</code>	Specifies the logging option [submit publish trace] (see page 243).
<code>-logpath dirPath</code>	Specifies a log directory path (see page 243).
<code>-t tempFilePath</code>	Specifies a path for temporary file created during deployment (needs space for up to ~5 Mb).
<code>deployment_name</code>	Name of the deployment to invoke (see page 253).

<code>[param=value]+</code>	Parameters that override configuration file parameters (for a full list of configuration file parameters, see “OpenDeploy Client Options” on page 261).
<code>-o package_name</code>	Creates a deployment package that can be transferred to the production server by alternate means (e.g., via email, manually, etc.). This option can only be used if TeamXpress-based comparison or a file list is used to determine which files to deploy.

Any configuration file parameter can be specified on the command line. These parameters will override parameters specified in the configuration file. In the case where parameters specified on the command line contradict parameters specified in the configuration file, the command-line parameters will be used.

Specifying Paths

UNIX

When specifying paths in UNIX, always use forward slashes.

Windows NT/2000

When specifying paths in the Windows NT/2000 Startup Parameters box, you can use either forward or back slashes. However, back slashes must be escaped by a preceding back slash. For example, the following path:

```
c:\iw-home\conf\iwodserver.cfg
```

will not work. Instead, specify all path names using one of the following conventions:

```
c:\\iw-home\\conf\\iwodserver.cfg
```

```
c:/iw-home/conf/iwodserver.cfg
```

When specifying paths in configuration files or at the Command Prompt, use single backslashes.

Options

Transactional Deployment

OpenDeploy's transactional deployment option allows you to ensure website integrity by making sure that if the deployment process is interrupted, the original website is preserved on the webserver.

To invoke transactional deployment, use the `-T` option when invoking the OpenDeploy client:

```
% iwdeploy -fs srcConfigFile -T deployment_name
```

Sequence of Events

The following events occur when you invoke transactional deployment:

1. The `iwdeploy` server is informed that it will be using transactional mode.
2. Changes are detected on the `iwdeploy` server side.
3. The `iwdeploy` server issues "get file" directives to the client.
4. The `iwdeploy` server records the filepaths in an internal file list.
5. The `iwdeploy` server makes copies of the original files:

```
cp file file.iwold
```
6. Upon receiving files, the `iwdeploy` server renames them with a `.iwnew` suffix. When all "gets" are done, three instances of every deployed file will exist: `*.iwold`, `*.iwnew`, and the original file.
7. The `iwdeploy` server renames the new files:

```
mv file.iwnew file'
```
8. The `iwdeploy` server deletes the old files:

```
rm *.iwold'
```

If any step of this process fails, the deployment is cancelled and the temporary files are removed. The original website files will be untouched.

Logging

OpenDeploy generates logging output for each deployment. You can specify the level of detail you want to include in a log, as well as what to name it and where to put it.

To configure OpenDeploy's logging option, invoke the `iwdeploy` client with the `-v`, `-log` and `-logpath` options:

```
% iwdeploy -fs srcConfigFile -V level -log option -logpath abs-path
deployment_name
```

Verbose Levels

The `-v` option allows you to specify the level of detail you want to include in a log. You can specify verbose logging levels from 1 to 4, where Level 1 is the least verbose, and Level 4 is the most (Level 4 only applies to the server log). See the log examples later in this chapter for examples of each level.

When a file or directory is deployed, the reason why it is deployed is included in the log:

<code>missing-in-dest</code>	For directory difference comparison, the element is not on the target server. For TeamXpress-based comparison, the element is not in the <code>previous_area</code> version path.
<code>missing-in-src</code>	For directory difference comparison, the element is not in the development server. For TeamXpress-based comparison, the element is not in the <code>area</code> version path.
<code>src-is-newer</code>	For directory difference comparison, the element is newer on the development server. For TeamXpress-based comparison, the element in the <code>area</code> <code>vpath</code> is newer.
<code>src-is-older</code>	Applies only when <code>revert</code> is specified. For directory difference comparison, the element is older on the development server. For TeamXpress-based comparison, the element in the <code>area</code> <code>vpath</code> is older.
<code>type-different</code>	Elements with the same name are of different types depending on where they reside. For example, the element found on one server or area is a file, while the element with the same path on the other server or area is a directory.
<code>user-different</code>	User is different (does not apply to Windows NT/2000).
<code>group-different</code>	Group is different (does not apply to Windows NT/2000).



mode-different	Permissions are different (does not apply to Windows NT/2000).
size-different	File size is different.
no-prev-area	TeamXpress-based comparison did not have <code>previous_area</code> specified.
file_list	The file was specified in a file list.

Here are some sample server and client logs, showing the level of detail for the `-v 1` through `-v 4` options. In the `-v 4` example, only the lines beginning with (2) are logged if you set the `-v 2` option, lines beginning with (2) and (3) are logged with the `-v 3` option, while all of the lines shown here are displayed if you specify the `-v 4` option. Unnumbered lines are logged in all cases.

-V 1

Server log

```

server: Waiting for connection...
server: Received connect request! (1)
Protocol Version(2.2) OK
  platform: server(UNIX), client(UNIX)
  Transaction Mode: OFF
  Mode(normal)
  Protocol(normal)
  Host(bogus)
Name(forward_deploy)
server: Number of local_directories to deploy: 1
server: Destination directory [/tmp/Branch1]
Options: do_deletes
server: COMPARE - dst[/tmp/Branch1] with
src[/u/iw/andre/deploytest/deploysrc/dir3]
server: Receiving item(/onedir)
server: Receiving item(/twodir)
server: Receiving item(/onedir/onedir.txt)
server: Receiving item(/twodir/twodir.txt)
Directories deployed : 2   Files deployed   : 4
Directories failed  : 0   Files failed   : 0
Directories deleted : 0   Files deleted   : 0
[Thu Apr 29 11:48:42 1999] Deployment COMPLETED

```

-V 4

```

(2) server: Bound to port 1709
server: Waiting for connection...
server: Received connect request! (1)
Protocol Version(2.2) OK
  platform: server(UNIX), client(UNIX)
  Transaction Mode: OFF
  Mode(normal)
  Protocol(normal)
  Host(bogus)
(2) server: Connection accepted!
  Name(forward_deploy)
server: Number of local_directories to deploy: 1
server: Destination directory [/tmp/Branch1]
Options: do_deletes
server: COMPARE - dst[/tmp/Branch1] with src[/u/iw/andre/deploytest/deploysrc/dir3]
(3) server: Getting directory info for (.)
(4) DIFF LEGEND <name type modDate user group mode size link>
DIFF src(onedir 3 925412751 2413:2200:777 512 )
  dst(onedir 3 925411586 0:1:40777 112 )
(4) DIFF LEGEND <name type modDate user group mode size link>
DIFF src(twodir 3 925412751 2413:2200:777 512 )
  dst(twodir 3 925411586 0:1:40777 112 )
(3) server: Getting directory info for (/onedir)
(4) DIFF LEGEND <name type modDate user group mode size link>
DIFF src(onedir.txt 1 924057137 2413:2200:640 0 )
  dst(onedir.txt 1 924057137 0:1:100640 0 )
(3) server: Getting directory info for (/twodir)
(4) DIFF LEGEND <name type modDate user group mode size link>
DIFF src(twodir.txt 1 924057147 2413:2200:640 0 )
  dst(twodir.txt 1 924057147 0:1:100640 0 )
(2) server: COMPARING done
(3) server: DEPLOYING to destination path [/tmp/Branch1]
(3) directive[reason src-is-newer]
(3) directive[get /onedir]
server: Receiving item(/onedir)
(3) server: dir for tempfile [/tmp/Branch1]
(3) directive[reason user-different]
(3) directive[reason src-is-newer]
(3) directive[get ./twodir]
server: Receiving item(/twodir)
(3) server: dir for tempfile [/tmp/Branch1]
(3) directive[reason user-different]
(3) directive[get /onedir/onedir.txt]
server: Receiving item(/onedir/onedir.txt)
(3) server: dir for tempfile [/tmp/Branch1/onedir]
(3) server: file created[/tmp/Branch1/onedir/onedir.txt-iwtmp]
(3) Cleaning: /tmp/Branch1/onedir/onedir.txt
(3) server: Renamed [/tmp/Branch1/onedir/onedir.txt-iwtmp] to
[/tmp/Branch1/onedir/onedir.txt]
(3) directive[reason user-different]
(3) directive[get ./twodir/twodir.txt]
server: Receiving item(/twodir/twodir.txt)
(3) server: dir for tempfile [/tmp/Branch1/twodir]
(3) server: file created[/tmp/Branch1/twodir/twodir.txt-iwtmp]
(3) Cleaning: /tmp/Branch1/twodir/twodir.txt
(3) server: Renamed [/tmp/Branch1/twodir/twodir.txt-iwtmp] to
[/tmp/Branch1/twodir/twodir.txt]
(3) directive[reason user-different]
Directories deployed : 2   Files deployed   : 4
Directories failed  : 0   Files failed   : 0
Directories deleted : 0   Files deleted   : 0
[Thu Apr 29 11:48:42 1999] Deployment COMPLETED

```



-V 1

Client log

```
[Thu Apr 29 12:06:21 1999] opendeploy 19990429.292.1 andre
INITIATED dir-diff forward_deploy sirius brain
Protocol Version(2.2) OK
  Transaction Mode: OFF
  Mode(normal)
  Protocol(normal)
  hostname(bogus)
  Name(forward_deploy)
client: Local_directories to deploy: 1
client: Options: do_deletes
client: DEPLOYING - [/u/iw/andre/deploytest/deploysrc/dir3] to
[/tmp/Branch1]
client: Sending [./one.txt] [reason src-is-newer] -- OK
client: Sending [./onedir] [reason user-different] -- OK
client: Sending [./two.txt] [reason src-is-newer] -- OK
client: Sending [./twodir] [reason user-different] -- OK
client: Sending [./onedir/onedir.txt] [reason user-different] -- OK
client: Sending [./twodir/twodir.txt] [reason user-different] -- OK
client: *** Note: UNENCRYPTED deployment was configured ***
client: Remote status: server-OK
[Thu Apr 29 12:06:22 1999] opendeploy 19990429.292.2 andre
COMPLETED /u/iw/andre/deploytest/deploysrc forward_deploy
sirius brain
[Thu Apr 29 12:06:22 1999] 19990429.292.2 STATS
DEPLOYMENT /u/iw/andre/deploytest/deploysrc --
forward_deploy OK
Directories deployed : 2 Files deployed : 4
Directories failed : 0 Files failed : 0
Directories deleted : 0 Files deleted : 0
```

-V 4

```
[Thu Apr 29 12:06:21 1999] opendeploy 19990429.292.1 andre
INITIATED dir-diff forward_deploy sirius brain
Protocol Version(2.2) OK
  Transaction Mode: OFF
  Mode(normal)
  Protocol(normal)
  hostname(bogus)
  Name(forward_deploy)
client: Local_directories to deploy: 1
client: Options: do_deletes
(3) client: COMPARE Phase
(2) client: Sending directory info for [.]
(2) client: Sending directory info for [./onedir]
(2) client: Sending directory info for [./twodir]
client: DEPLOYING - [/u/iw/andre/deploytest/deploysrc/dir3] to [/tmp/Branch1]
(3) client: Received 'get' request for (./one.txt)
client: Sending [./one.txt] [reason src-is-newer] -- OK
(3) client: Received 'get' request for (./onedir)
client: Sending [./onedir] [reason user-different] -- OK
(3) client: Received 'get' request for (./two.txt)
client: Sending [./two.txt] [reason src-is-newer] -- OK
(3) client: Received 'get' request for (./twodir)
client: Sending [./twodir] [reason user-different] -- OK
(3) client: Received 'get' request for (./onedir/onedir.txt)
client: Sending [./onedir/onedir.txt] [reason user-different] -- OK
(3) client: Received 'get' request for (./twodir/twodir.txt)
client: Sending [./twodir/twodir.txt] [reason user-different] -- OK
client: *** Note: UNENCRYPTED deployment was configured ***
client: Remote status: server-OK
[Thu Apr 29 12:06:22 1999] opendeploy 19990429.292.2 andre
COMPLETED /u/iw/andre/deploytest/deploysrc forward_deploy sirius
brain
[Thu Apr 29 12:06:22 1999] 19990429.292.2 STATS DEPLOYMENT
/u/iw/andre/deploytest/deploysrc -- forward_deploy OK
Directories deployed : 2 Files deployed : 4
Directories failed : 0 Files failed : 0
Directories deleted : 0 Files deleted : 0
```


Log Names and Locations

The `-log` option has three possible values: `trace`, `submit`, or `publish`. If no option is specified, the log data will be sent to `stdout`. This option only applies to the OpenDeploy client. Under UNIX, the server-side log file is located in `opendeploy/iwdeploy.log`. Under Windows NT/2000, a server-side log file is automatically generated in the location you specify at the time of installation.

The `trace` option creates the following log file under UNIX:

```
MODETraceDATE.tag.log
```

or, under Windows NT/2000:

```
MODETraceDATE.tag.txt
```

The `submit` option creates the following log file under UNIX:

```
MODESubmitDATE.tag.log
```

or, under Windows NT/2000:

```
MODESubmitDATE.tag.txt
```

The `publish` option creates the following log file under UNIX:

```
MODEPublishEDITIONDATE.tag.log
```

or, under Windows NT/2000:

```
MODEPublishEDITIONDATE.tag.txt
```

where:

MODE = {deploy|reverse}

EDITION = [edition-name]

DATE = [yyyymmdd]

tag = the OpenDeploy session tag

For example, if `iwdeploy` were called by the TeamXpress command line tool `iwatsub`, you would use the `-log submit` option. If `iwdeploy` were called by the TeamXpress command line tool `iwatpub`, you would use the `-log publish` option. A timed deployment might use the `-log trace` option. These naming options are for convenience in identifying the deployment trigger only—they do not affect the contents of the logs.

On Windows NT/2000, each deployment session creates a new trace log file. Script output is stored in a separate trace log file from the server trace log.

To specify the location of the log file, use the `-logpath` option. The specified path must be an absolute path. The default path is:

`iw-home/log`

If `iw-home` cannot be found, the default path under UNIX will be:

`/var/adm`

or, under Windows NT/2000:

`C:\installation directory\OpenDeploy\log`

Event Reporting

If you are using OpenDeploy in conjunction with TeamXpress, you can integrate OpenDeploy logging with the TeamXpress Global Report Center. To activate this option, specify the `-events` tag when you invoke the client for each deployment you want recorded in the Global Report Center:

```
% iwdeploy -fs srcConfigFile -events deployment_name
```

The log data that will be fed into the reporting system will be generated in a `deployEvents.log` file.



Chapter 14

Configuration Files

Configuration files for `iwdeploy` specify all the options needed for deployment. At least two such files, one for client configuration and one for the server, are required for any deployment scenario. Some scenarios require multiple client and server configuration files (see Chapter 17, “Deployment Scenarios,” for examples of configuration files for common scenarios). Typical options to set in client and server configuration files include source and destination directories, which files to exclude from deployment, what permissions to set on deployed files, and many others (see Chapter 15, “Configuration File Options”).

Configuration files for `iwdeploy` (both client and server) are exactly the same under both UNIX and Windows NT/2000, except for pathnames, which must be specified according to platform.

OpenDeploy Server Configuration Files

The structure of the server configuration file is a shallow hierarchy of sections. Sections are delimited by an opening keyword (e.g. `TeamSite_server=name`, which specifies the beginning of a `TeamSite_server` section) and a closing `;` on a line by itself. A server configuration file may contain one or more named `deployment` sections. Each `deployment` section contains an `allowed_directory` section and may contain a `deploy_run_script` section. Lines containing comments can appear anywhere in the configuration file.

The default server configuration file is located in `opendeploy\conf\iw.odserver.cfg` for UNIX or `C:\Program Files\Interwoven\OpenDeploy\conf\iwodserver.cfg` for Windows NT/2000. The following examples illustrate the structure of OpenDeploy server configuration files.

UNIX

```
port=1701

TeamSite_server=development1.example.com
  allowed_directory=/local/andre/deploydir
  key_file=/u/iw/andre/secret_file.txt
  deployment=deployandre
    client_is_trusted=no
    allowed_directory=/usr/local/etc/httpd/htdocs
    exclude_pattern=script_one
  ;
  deploy_run_script=script_one
  as=andre
  when=server_before_deploy
  where=/home/andre
;
;
```

Windows NT/2000

```
port=1701

TeamSite_server=development1.example.com
  key_file=d:\deploy\encryptkey
  allowed_directory=d:\deploydst1\content
  deployment=deployandre
    client_is_trusted=no
    allowed_directory=d:\deploydst2\content
    exclude_pattern=script_one
  ;
  deploy_run_script=script_one
  as=andre
  when=server_before_deploy
  where=d:\deploydst3\content
;
;
```

Chapter 15, “Configuration File Options,” contains a full list of `iwdeploy` server configuration options.

Global options

TeamXpress server

section: specifies

Deployment

section: specifies

Deploy-Run-Script

section: specifies

Global options

TeamXpress server

section: specifies

Deployment

section: specifies

Deploy-Run-Script

section: specifies

OpenDeploy Client Configuration Files

The structure of the client configuration file is a shallow hierarchy of sections. Sections are delimited by an opening keyword (e.g. `deployment=name`, which specifies the beginning of a `deployment` section) and a closing `;` on a line by itself. A client configuration file contains one or more named `deployment` sections. Each deployment section contains one or more `local_directory` sections. Lines containing comments can appear anywhere in the configuration file. While there is no default client configuration file, OpenDeploy comes with a sample file (`/conf/iw.odclient.example` or `C:\Program Files\Interwoven\OpenDeploy\conf\iwodclient`) that can be a useful starting point.

Here is a sample client configuration file that corresponds to the server configuration file shown earlier in this chapter:

<code>hostname=development1.example.com</code>	Global
<code>source_exclude_pattern=~\$</code>	
<code>key_file=/u/iw/andre/secret_file.txt</code>	
<code>deployment=deployandre</code>	Deployment
<code>remote_server=production1.example.com</code>	
<code>remote_port=1701</code>	
<code>area>//IWSERVER/default/main/dev/WORKAREA/andre</code>	
<code>do_deletes</code>	
<code>local_directory=html</code>	Local
<code>remote_directory=/usr/local/etc/httpd/htdocs</code>	
<code>source_exclude=test</code>	
<code>exclude=log</code>	
<code>;</code>	
<code>local_directory=cgi-bin</code>	Local
<code>remote_directory=/usr/local/scripts/cgi-bin</code>	
<code>source_exclude=test</code>	
<code>exclude=log</code>	
<code>;</code>	
<code>;</code>	
<code>deployment=deploychris</code>	Deployment
<code>remote_server=production2.example.com</code>	
<code>remote_port=1701</code>	
<code>area>//IWSERVER/default/main/dev/WORKAREA/chris</code>	



```

local_directory=html
    remote_directory=/usr/local/etc/httpd/htdocs
    exclude=log
;
;

```

Local

This configuration file contains two deployment sections, `deployandre` and `deploychris`. The `deployandre` deployment section contains two `local_directory` sections and the `deploychris` section one. Each deployment section corresponds to a single set of transfer operations to a single machine. If you were to invoke the `iwdeploy` client as

```
% iwdeploy deployandre
```

with the preceding configuration file, the `iwdeploy` client would transfer the files and directories in the `html` directory of `workarea andre`¹ to `/usr/local/etc/httpd/htdocs` and the `cgi-bin` directory to `/usr/local/scripts/cgi-bin` on `production1.example.com`.

Chapter 15, “Configuration File Options,” contains a full list of the `iwdeploy` client configuration options.

Coordinating Server and Client Configuration Files

Several client and server options correspond to each other, and must specify the same value for a deployment to proceed:

Server option	Client option	Description
<code>port</code>	<code>remote_port</code>	Port to listen on/port to send to.
<code>TeamSite_server</code>	<code>hostname</code>	The server on which the OpenDeploy client resides.
<code>key_file</code>	<code>key_file</code>	File to be used in encryption.
<code>deployment</code>	<code>deployment</code>	Section of the client and server configuration files to use (only required on the server if <code>client_is_trusted=no</code> is specified).

1. TeamXpress areas are specified using version paths, or vpaths. For a full explanation of vpaths, consult *Administering TeamXpress*.

Server option	Client option	Description
<code>allowed_directory</code>	<code>remote_directory</code>	Directory on the production server to which content is to be deployed.

Scope of Configuration File Options

The options specified in the configuration files apply only to particular sections. For example, in the client configuration file, the `do_deletes` directive is only used by `local_directory` sections. However, where you specify `do_deletes` controls which `local_directory` sections it applies to. You can specify `do_deletes` (and most other options) in an enclosing section and have it apply to all following `local_directory` sections.

Global options must be specified at the global level. Deployment options are normally specified in a `deployment` section, in which case they will apply only to that deployment, but they can be also specified at the global level, in which case they will apply to all succeeding deployments unless contradicted at a lower level. On the client, local directory options are normally specified at the `local_directory` level, in which case they will apply only to the local directory, but they can also be specified at the `deployment` level, in which case they will apply to all local directories in that deployment section unless contradicted at the `local_directory` level. Local directory options can even be specified at the global level, in which they will apply to all succeeding deployments unless contradicted at a lower level.

The only exception to these scoping rules is Deploy and Run. All Deploy and Run options must be contained within a `deploy_run_script` section, which must be specified at the local directory level or, on the server, at the deployment level.

The examples on the following two pages show how the level at which an option is specified affects the behavior of the deployment.



For example, if your configuration file contains:

```
...
host=development1.example.com
deployment=andre
    remote_server=production1.example.com
    area=//IWSERVER/default/main/dev/WORKAREA/andre
    local_directory=demo
        remote_directory=/stuff/demo
        do_deletes
    ;
    local_directory=test
        remote_directory=/stuff2/test
    ;
;
```

then `do_deletes` applies to the `demo local_directory` section, but not to the `test local_directory` section. If instead you have:

```
...
host=development1.example.com
deployment=andre
    remote_server=production1.example.com
    area=//IWSERVER/default/main/dev/WORKAREA/andre
    do_deletes
    local_directory=demo
        remote_directory=/stuff/demo
    ;
    local_directory=test
        remote_directory=/stuff2/test
    ;
;
```

then `do_deletes` applies to both `local_directory` sections. It would not apply to any deployment sections which followed the `andre` section.

If you have:

```
...
hostname=development1.example.com
do_deletes
deployment=andre
    remote_server=production1.example.com
    area=//IWSERVER/default/main/dev/WORKAREA/andre
    local_directory=demo
        remote_directory=/stuff/demo
    ;
    local_directory=test
        remote_directory=/stuff2/test
    ;
;
...
```

then `do_deletes` would apply to the `andre` deployment section and all following deployment sections, but not to any deployment sections which came before it. Finally, if you have:

```
...
host=development1.example.com
deployment=andre
    remote_server=production1.example.com
    area=//IWSERVER/default/main/dev/WORKAREA/andre
    do_deletes
    local_directory=test
        remote_directory=/stuff2/test
        dont_do
    ;
    local_directory=demo
        remote_directory=/stuff/demo
    ;
;
```

then `do_deletes` would apply to `demo`, but not to `test` because `test` contains the local directory option `dont_do`, which directly contradicts `do_deletes`.

Options specified on the command line behave as if they were options specified at the beginning of the configuration file (i.e., at the global level).

Use of Client versus Server Configuration Options

Several configuration options can be specified in either the client or the server configuration files. The behavior of these options depends on how the `client_is_trusted` option is specified. If the client is trusted, then the client configuration file options are used. The server configuration options are used only if they do not contradict the options specified on the client.

If the client is not trusted, then the server-side options override all options specified on the client. The only client options that get used are the name of the deployment section to deploy to, which content on the client-side to deploy, and any other specifications that are processed on the client side only. For example, the `source_exclude` option applies exclusively to the OpenDeploy client, so it cannot be overridden by any option specified on the OpenDeploy server. The `remote_directory` client option will also be used if it falls under an `allowed_directory` (as specified on the server).

The Authorization Configuration File

The authorization configuration file allows you to specify which users and groups can perform a particular named deployment to a particular destination server. This file exists on the destination server that it applies to, and it applies only when `client_is_trusted=no` is specified in the server configuration file.

This configuration file contains any number of lines in the following format:

```
deployment=allowed_list
```

where *deployment* is the name of the deployment specified in the client and server configuration files, and *allowed_list* is a comma-separated list of allowed users and groups. For example, an authorization configuration file might contain the following lines:

```
deployUNIX=chris, andre, tsusers  
deploytest=andre
```

or, for Windows NT/2000:

```
deployWinNT=EXAMPLE\chris, EXAMPLE\andre, EXAMPLE\tsusers  
deploytest=EXAMPLE\andre
```

Note that on Windows NT/2000, users must be specified with domain names.

If `client_is_trusted=no` is specified in the server configuration file, and a named deployment is invoked by a user who is not authorized to do so, it will fail. If `client_is_trusted=no` is specified in the server configuration file, and a named deployment that is not specified in this configuration file is invoked, it will fail.

To pass the authorization file to the OpenDeploy server, use the `-auth` command line option.



Chapter 15

Configuration File Options

OpenDeploy configuration options are specified in the client and server configuration files. Most options are specified in the client configuration file, or at the command line when invoking the client. These are described in the following section. Options specified in the server configuration file are described in “OpenDeploy Server Options” on page 289.

OpenDeploy Client Options

The client options allow you to specify the following:

- Deployment sections (see page 262)
- Deployment targets (see page 262)
- Locations of files to be deployed (see page 263)
- Which files to deploy (see page 265)
- Which files to exclude (see page 275)
- Which files to rename or delete during deployment (see page 280)
- Changes to file permissions during deployment (see page 281)
- Encryption (see page 285)
- Deploy and Run (see page 286)
- Links handling (see page 288)
- Deployment configuration debugging (see page 288)

Note that the behavior of many configuration options depends entirely on how the `client_is_trusted` option is specified on the server. If the client is not trusted, then the server-side options override all options specified on the client. The only client options that get used are the name of the deployment section to deploy to, which content on the client-side to deploy, and any other specifications that are processed on the client side only. The `remote_directory` client option will also be used if it falls under an `allowed_directory` (as specified on the server).

If the client is trusted, then its options are used. The only server options that are used (of the options that can be specified on either client or server) are the ones that do not contradict the options specified on the client.

Specifying Deployment Sections

A single client configuration file can be used to invoke several different types of deployment. Each different type of deployment can be independently configured in deployment sections. Each configuration file must have at least one deployment section. To specify a new deployment section, use the `deployment=name` option.

`deployment=name`

This is a global option, which begins a deployment section named *name*. The name of the section is used in the command line for the `iwdeploy` client.

Specifying Deployment Targets

The deployment options `remote_port` and `remote_server` specify information about the destination server for deployment. Both of these options are required.

`remote_port=#`

This specifies the port on which the `iwdeploy` server on the destination server is listening, e.g.:

`remote_port=1709`

`remote_server=server`

This specifies the name of the destination server, e.g.:

`remote_server=production1.example.com`


```
remote_directory=absolute_path
```

The `remote_directory` option applies to `local_directory` sections, although it may be specified at a higher level in order to apply to multiple sections. The remote directory must correspond to the `allowed_directory` specified in the server configuration file, or to one of its subdirectories. This option is required, and must specify the *absolute* path on the destination server to which this `local_directory` section should be deployed, e.g.:

```
local_directory=deploysrc
    remote_directory=d:\deploydst\content
```

Specifying Deployment Timeouts

```
timeout=#seconds
```

This option specifies the number of seconds it will take for the OpenDeploy client process to time out. `timeout` is a deployment option, but it can also be specified at a higher level.

By default, OpenDeploy will time out at 150 seconds. However, if you are doing a TeamXpress comparison-based deployment, you may need to specify a larger number so that OpenDeploy does not time out before the comparison is completed. For example:

```
timeout=25000
```

Specifying Locations of Files to Be Deployed

```
hostname=name
```

This is a global option that identifies the sending server to the `iwdeploy` server. This option is required, e.g.:

```
hostname=development1.example.com
```

`area=path`

This specifies the area from which all `local_directory` sections are based. The path can be a version path (vpath)¹ of the form `//IWSERVER/...` or it can be an absolute path. To specify the next-to-last edition on a branch, use a vpath ending in `/EDITION/IW_PREV`. To specify the latest edition on a branch, use a vpath ending in `/EDITION`. For example:

UNIX absolute path:

```
area=/iwmnt/default/main/dev/EDITION
```

Windows NT/2000 absolute path:

```
area=y:\default\main\dev\EDITION
```

Version path:

```
area>//IWSERVER/default/main/dev/EDITION
```

`local_directory=path`

This begins a new `local_directory` section. The path specifies a path *relative* to the enclosing area from which files and directories should be deployed. For example:

UNIX:

```
local_directory=htdocs/gifs
```

1. For more information on version paths, see *TeamXpress Command-Line Tools*. If you are not using TeamXpress, you cannot use version paths.

Windows NT/2000:

```
local_directory=htdocs\gifs
```

A configuration file with the following two lines specifies a section that would apply to the `deploysrc` directory in the most recent edition on the `dev` branch:

```
area=y:\default\main\dev\EDITION  
    local_directory=deploysrc
```

All lines contained in this section would apply only to this directory.

Specifying Which Files to Deploy

OpenDeploy can use one of three methods to determine which files to deploy:

- Directory comparison
- TeamXpress comparison
- File lists













Directory Comparison

Directory comparison is the default option. It compares the directory being deployed from with the directory being deployed to. By default, it deploys the files in the directory being deployed from that are newer than the corresponding files in the directory being deployed to. However, directory comparison can also use the `revert` option to deploy the older versions of files (to revert to a previous version of the website), or it can use the `date_different` option to deploy files that have any difference in the date. This option can be specified for an entire configuration file, an entire named deployment section, or a specific local directory.















Default Directory Comparison

With the default option (`date_different` or `revert` is not specified), only newer files or files whose size or other attributes are different will be deployed:

Source webserver (OpenDeploy client)	Destination webserver (OpenDeploy server)	Action
 file1 9/21/98 3:42 PM 23K	 file1 9/24/98 3:16 PM 23K	not deployed (source is older)
 file2 9/23/98 4:27 PM 98K	 file2 9/23/98 4:27 PM 98K	not deployed (same date, same size)
 file 3 9/28/98 9:11 AM 35K	 file 3 7/18/98 4:32 PM 37K	deployed (source is newer)
 file 4 9/30/98 1:56 PM 56K	 file 4 9/30/98 1:56 PM 32K	deployed (same date, different size)
 file 5 9/27/98 4:38 PM 56K	 file 5 N/A	deployed (destination file does not exist)
 file 6 N/A	 file 6 9/30/98 2:20 PM 56K	File does not exist on source webserver; if <code>do_deletes</code> is specified, the destination file will be deleted. Otherwise, the destination file will be ignored.

The Revert Option

When `revert` is specified, only older files or files whose size has been changed will be deployed:

Source webserver (OpenDeploy client)	Destination webserver (OpenDeploy server)	Action
 file1 9/21/98 3:42 PM 23K	 file1 9/24/98 3:16 PM 23K	deployed (source is older)
 file2 9/23/98 4:27 PM 98K	 file2 9/23/98 4:27 PM 98K	not deployed (same date, same size)
 file 3 9/28/98 9:11 AM 35K	 file 3 7/18/98 4:32 PM 37K	not deployed (source is newer)
 file 4 9/30/98 1:56 PM 56K	 file 4 9/30/98 1:56 PM 32K	deployed (same date, different size)
 file 5 9/27/98 4:38 PM 56K	 file 5 N/A	deployed (destination file does not exist)
 file 6 N/A	 file 6 9/30/98 2:20 PM 56K	File does not exist on source webserver; if <code>do_deletes</code> is specified, the destination file will be deleted. Otherwise, the destination file will be ignored.

To revert a website, specify the `revert` option in the configuration file:

`revert`

This specifies that only files that are older on the source should be transferred to the destination. By default files are transferred only if they are newer. This option applies to `local_directory` sections, although it can be specified at a higher level so that it applies to multiple sections.



The Date-Different Option

The Date-Different option allows you to deploy any files with differences in date or size:

Source webserver (OpenDeploy client)	Destination webserver (OpenDeploy server)	Action
file1 9/21/98 3:42 PM 23K	file1 9/24/98 3:16 PM 23K	deployed (destination date is different)
file2 9/23/98 4:27 PM 98K	file2 9/23/98 4:27 PM 98K	not deployed (same date, same size)
file 3 9/28/98 9:11 AM 35K	file 3 7/18/98 4:32 PM 37K	deployed (destination date is different)
file 4 9/30/98 1:56 PM 56K	file 4 9/30/98 1:56 PM 32K	deployed (same date, different size)
file 5 9/27/98 4:38 PM 56K	file 5 N/A	deployed (destination file does not exist)
file 6 N/A	file 6 9/30/98 2:20 PM 56K	File does not exist on source webserver; if <code>do_deletes</code> is specified, the destination file will be deleted. Otherwise, the destination file will be ignored.

To use this option, specify `date_different` in the client configuration file.

`date_different`

If this option is specified, files will be transferred from the source to the destination if the modification dates of corresponding files are different. The default option is for files to be transferred only if the modification date of the source file is newer than the modification date of the destination file.

This option applies to `local_directory` sections, although it can be specified at a higher level so that it applies to multiple sections.

Examples

This simple client configuration file uses the default deployment option of directory comparison. It deploys files from the TeamXpress host `development1`. Encryption is not being used. It contains one named deployment section, `default`, which deploys files to `production1` on port 1701. Files are to be deployed from the latest edition on the `dev` branch, and all files contained within that branch are deployed. The directory on `production1` that files are to be deployed to is `/usr/local/etc/httpd/htdocs`. If a file has been deleted from the area being deployed, the corresponding file will be deleted on the destination server.

```
hostname=development1.example.com
deployment=default
    remote_server=production1.example.com
    remote_port=1701
    area=//IWSERVER/default/main/dev/EDITION
    local_directory=.
        remote_directory=/usr/local/etc/httpd/htdocs
        do_deletes
    ;
;
```

The deployment name is `default`, and the configuration file is in the default location (`/etc/iw.deploy.cfg`) so to invoke this deployment you would type (on the source system):

```
% iwdeploy default
```

You can specify any configuration file option on the command line, so to use the OpenDeploy Site Rollback option without altering the configuration file you would type:

```
% iwdeploy default revert
```

or, to use the date-different option you would type:

```
% iwdeploy default date_different
```



To specify the date-different or the revert option in the configuration file, add the option to the section you want to use this type of deployment in. If you want to use the option for all deployments, specify it at the global level. To use it for a particular named deployment, specify it in the named deployment's section of the configuration file (as shown below). To use it for a particular directory being deployed, specify it in that local directory section of the named deployment:

```
hostname=development1.example.com
deployment=datediff
    remote_server=production1.example.com
    remote_port=1701
    area=//IWSERVER/default/main/dev/EDITION
    date_different
    local_directory=.
        remote_directory=/usr/local/etc/httpd/htdocs
        do_deletes
    ;
;
```

TeamXpress Comparison

TeamXpress comparison uses the TeamXpress Compare feature to compare any two TeamXpress areas and deploy the differences. To use this type of comparison, add the `TeamSite_based` and `previous_area` options to the client configuration file. OpenDeploy will compare `area` (see page 264) with `previous_area`, and deploy the results.

TeamSite_based

This option compares `area` and `previous_area` and uses the output of comparison for deployment. For example:

- Compare a TeamXpress workarea with the staging area and deploy all content modified in the workarea to the destination server, or
- Compare two editions and deploy incremental changes

This option is specified at the deployment level, and it requires the use of the `previous_area` option.

`previous_area=path`

This option specifies the TeamXpress area to compare `area` with. It is a deployment-level option, and it requires `area` to be a TeamXpress area, and for the `TeamSite_based` option to be specified. `previous_area` is specified in the same manner as `area`:

UNIX absolute path:

```
previous_area=/iwmnt/default/main/dev/IW_PREV
```

Windows NT/2000 absolute path:

```
previous_area=y:\default\main\dev\IW_PREV
```

Version path:¹

```
previous_area=//IWSERVER/default/main/dev/IW_PREV
```

This simple deployment configuration file is the same as the Directory Comparison example (see page 269), except that it uses TeamXpress comparison to compare the most recent edition on a branch with the next most recent:

```
hostname=development1.example.com
deployment=TeamSitecompare
    remote_server=production1.example.com
    remote_port=1701
    TeamSite_based
    area=//IWSERVER/default/main/dev/EDITION
    previous_area=//IWSERVER/default/main/dev/EDITION/
    IW_PREV
    local_directory=.
        remote_directory=/usr/local/etc/httpd/htdocs
        do_deletes
    ;
;
```

1. For more information on vpaths, or version paths, consult *Administering TeamXpress*.

To invoke this deployment, you would type (on the source server):

```
% iwdeploy TeamSitecompare
```

Alternatively, you could specify the `TeamSite_based` and `previous_area` options on the command line. This command, with the configuration file on page 269, would produce the same results as the example above.

```
% iwdeploy default TeamSite_based previous_area=//IWSERVER/default/main/  
dev/EDITION/IW_PREV
```

File Lists

OpenDeploy can also deploy a list of files. This list can be static, or it can be dynamically generated (e.g. using the `iwevents` command line tool). The list of files is contained in a file that is specified by the `file_list` option. For information on using the `do_deletes` option in conjunction with the `file_list` option, also see “File List With Deletions” on page 274.

A typical deployment list looks like:

```
/www/index.html  
/www/andre/index.html  
/www/products.html
```

where `/www` is a directory immediately subordinate to the `area` root directory.

To deploy a list of files, specify the `file_list` option in the OpenDeploy client configuration file:

```
file_list=path
```

This local directory option specifies a path to a file containing a list of paths or version paths to individual files to be deployed, one file to a line (see above). For named deployments using `file_list`, only one local directory can be specified. The version paths of the individual files must be relative to the local directory specified. For example, if the `local_directory` is `/www`, the version paths listed in the file list would be relative to `/www`. In addition, if you specify the `file_list` option at the command line, you can stream a list of files for deployment in from `stdin` (see the example above). UNIX and Windows NT/2000 examples are as follows:

```
file_list=/tmp/andre_deploy_list
file_list=C:\deploy\andre_deploy_list
```

This simple deployment configuration file is the same as the examples given in the “Directory Comparison” and “TeamXpress Comparison” sections, except that it deploys a list of files. In this case, the file is streamed in from `stdin`:

```
hostname=development1.example.com
deployment=filelist-1
    remote_server=production1.example.com
    remote_port=1701
    file_list=-
    area=//IWSERVER/default/main/dev/EDITION
    local_directory=.
        remote_directory=/usr/local/etc/httpd/htdocs
    ;
;
```

This deployment would be invoked by the command:

```
% iwdeploy filelist-1 < /tmp/andre_deploy_list
```

where `/tmp/andre_deploy_list` is the file containing the list of files to be deployed.

This deployment configuration file is the same as the example above, except that the file list is specified in the configuration file:

```
hostname=development1.example.com
deployment=filelist-2
    remote_server=production1.example.com
    remote_port=1701
    file_list=/tmp/andre_deploy_list
    area=//IWSERVER/default/main/dev/EDITION
    local_directory=.
        remote_directory=/usr/local/etc/httpd/htdocs
;
;
```

To invoke this deployment, you would type (on the source server):

```
% iwdeploy filelist-2
```

File List With Deletions

If `file_list` is specified in the configuration file, then if the files referenced in the file list do not appear on the source server, but they do appear on the destination server, the files on the destination server will remain intact.

If `do_deletes` is specified in addition to `file_list` is specified in the configuration file, then files referenced in the file list that do not appear on the source server, but which do appear on the destination server, will be deleted on the destination server.

The following example illustrates the difference between these two modes. Letters A–E represent the files that are listed in the file list, or that are present on the source or destination servers. A plus sign (+) indicates that the file has been modified.

File List	Source	Destination	Result with <code>file_list</code> option alone	Result with <code>do_deletes</code> option
A	A+	A	A+ is sent to destination	A+ is sent to destination
B	B	B	B is sent to destination	B is sent to destination
C	C+		C+ is created in destination	C+ is created in destination

	D		D is ignored	D is ignored
E		E	E is ignored	E is deleted from destination

The use of `file_list` and `do_deletes` with directories introduces some additional subtleties:

File List	Source	Destination	Result with <code>file_list</code> option alone	Result with <code>do_deletes</code> option
/DirA	/DirA/fileA+	/DirA/fileA	/DirA/fileA+ is sent to destination	/DirA/fileA+ is sent to destination
	/DirA/fileB	/DirA/fileB	/DirA/fileB is sent to destination	/DirA/fileB is sent to destination
		/DirA/fileG	/DirA/fileG is ignored in destination	/DirA/fileG is ignored in destination ¹
/DirB/fileC	/DirB/fileC		/DirB is created in remote dir, /DirB/fileC is created in destination	/DirB is created in remote dir, /DirB/fileC is created in destination
	/DirB/fileD		/DirB/fileD is ignored	/DirB/fileD is ignored
/DirC		/DirC/fileE	/DirC is ignored, /DirC/fileE is ignored	/DirC is deleted in destination, /DirC/fileE is deleted
		/DirC/fileF	/DirC/fileF is ignored	/DirC/fileE is deleted

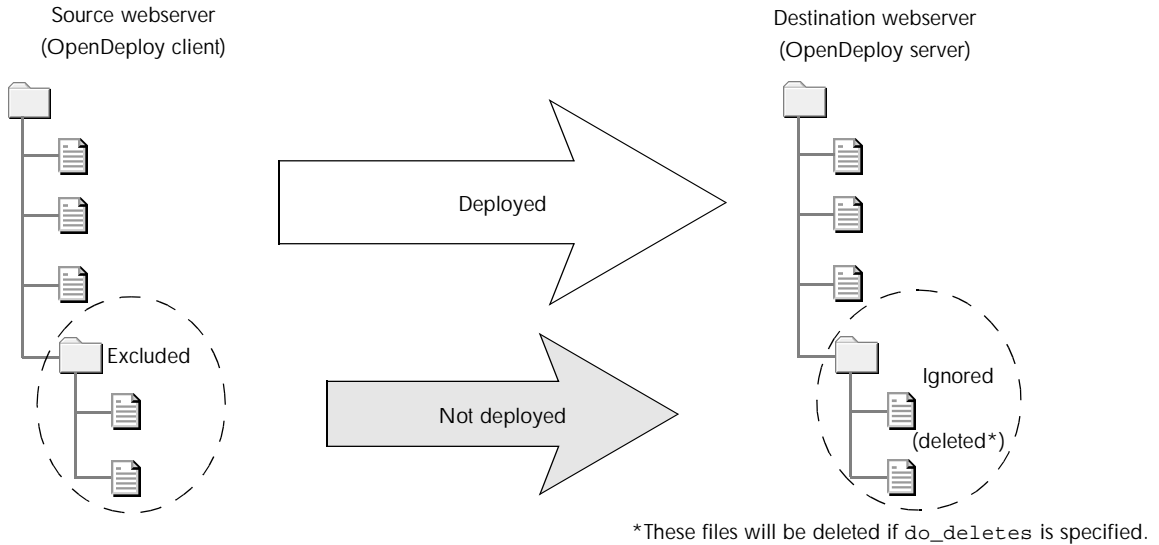
1. /DirA/fileG is still ignored because it is not specified in `file_list`

Specifying Which Files to Exclude

These options allow you to specify particular files or directories not to deploy. All of these options apply to `local_directory` sections, although they can be specified at a higher level to apply to multiple sections. The options, described later in this section, allow for three main types of exclusion: on the source server, on the destination server, and on both servers.

Excluding Files on the Source

When a file or directory is excluded from deployment, it is treated as though it does not exist. If a file or directory is excluded only on the source side, and if `do_deletes` is specified (see page 280), the corresponding file or directory (if any) on the destination side will be deleted. If `do_deletes` is not specified, the corresponding file or directory on the destination side will be ignored. For example:



Files excluded on the source server

In the example above, files are excluded only on the source server. Any corresponding files on the destination server will be ignored in the deployment process, unless `do_deletes` is specified. If `do_deletes` is specified, the corresponding files will be deleted.

To exclude files on the source side, use the `source_exclude` and `source_exclude_pattern` options. Both of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified on the server (see page 290).

`source_exclude=path`

This specifies a path *relative* to the `local_directory` path that should be excluded from deployment. The effect of this is as if the specified path did not exist on the source. Any number of `source_exclude` options can be specified.

`source_exclude_pattern=pattern`

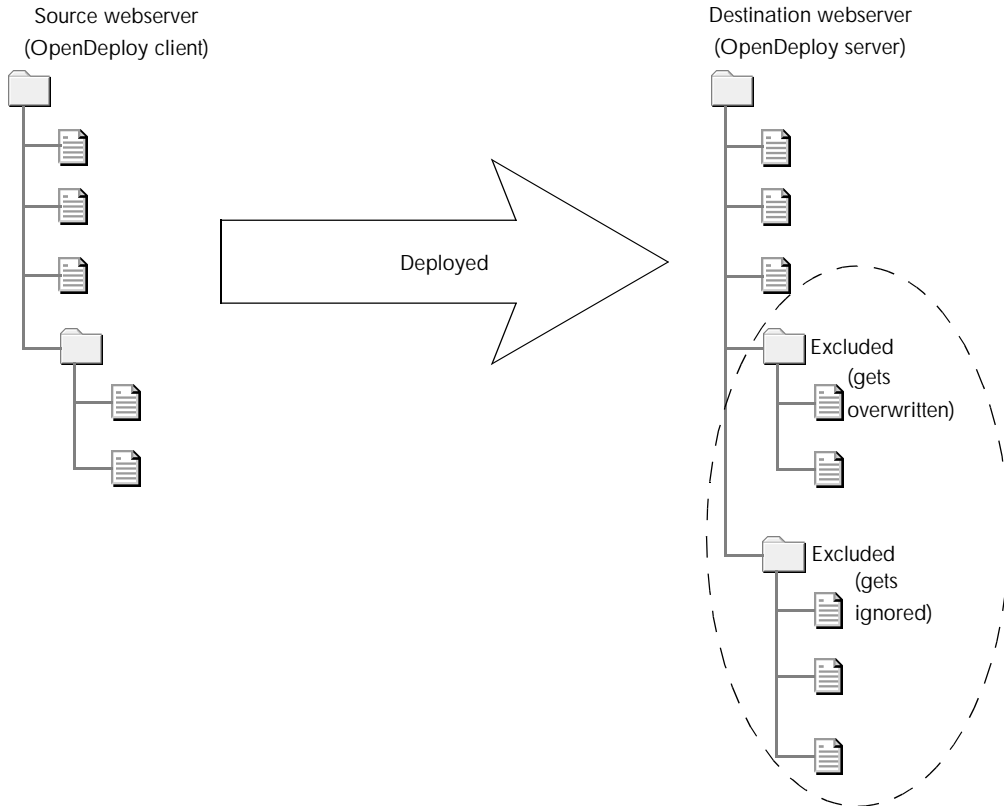
This specifies a regular expression pattern to exclude on the source. The syntax of the patterns is `regex(5)` (extended syntax). The items actually compared to the pattern are paths relative to the `local_directory`. The paths always begin with `./`. Any number of `source_exclude_pattern` options can be specified. For example:

```
source_exclude_pattern="/htdocs/company/*.html"
```

would exclude all items under subdirectory `/htdocs/company` that end in `html`.

Excluding Files on the Destination Server

If a file or directory is excluded only on the destination side, and a corresponding file or directory is deployed from the source, the existing file or directory on the destination side will be overwritten. For example:



Files excluded on the destination server

In the example above, files are excluded only on the destination server. Their status will be ignored in the deployment process. If corresponding files exist on the source server, the excluded files will be overwritten on the destination server. If no corresponding files exist on the source server, the excluded files on the destination server will be ignored.

To exclude files on the destination server, use the `destination_exclude` and `destination_exclude_pattern` options. Both of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`destination_exclude=path`

This specifies a path *relative* to the `remote_directory` path which should be excluded from deployment. The effect of this is as if the specified path did not exist at the destination. If there is a corresponding path on the source side, the destination side will be unconditionally overwritten. Any number of `destination_exclude` options can be specified.

`destination_exclude_pattern=pattern`

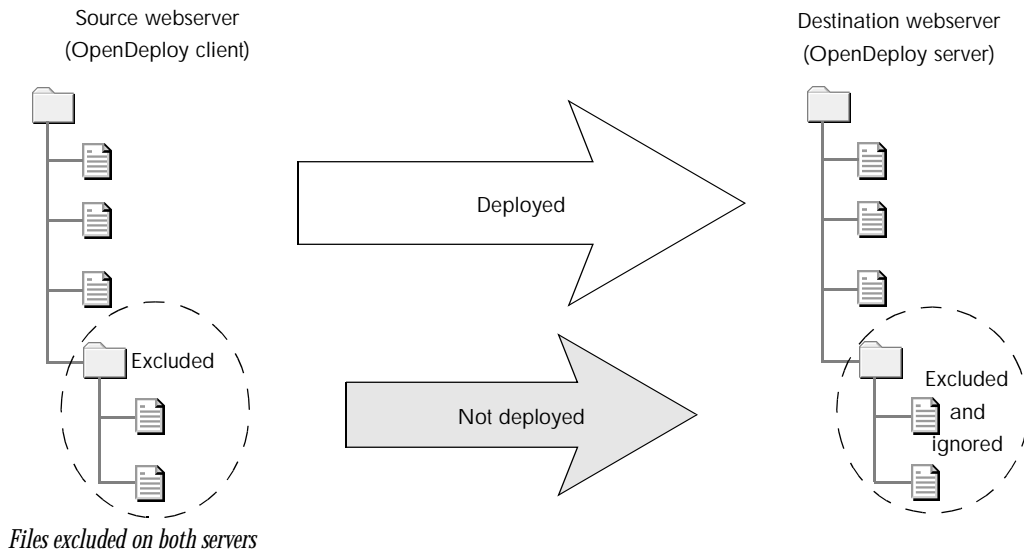
This specifies a regular expression pattern to exclude at the destination. The syntax of the patterns is `regex(5)` (extended syntax). The items actually compared to the pattern are paths relative to the `remote_directory`. The paths always begin with `./`. Any number of `destination_exclude_pattern` options can be specified. For example:

```
destination_exclude_pattern="^/htdocs/company/*.html"
```

would exclude all items under subdirectory `/htdocs/company` that end in `html`.

Excluding Files on Both Servers

If a file or directory is excluded from both the source and destination, the source file will not be deployed and the destination file will not be overwritten. For example:



In this example, files are excluded on both the source and the destination servers. During deployment, files that are excluded on both servers are ignored and not deleted.

To exclude files on both servers, use the `exclude` and `exclude_pattern` options. All of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`exclude=path`

This specifies a path that should be excluded from both the source and destination. The path is *relative* to the `local_directory` specification on the source and the `remote_directory` specification on the destination server.

`exclude_pattern=pattern`

This specifies a regular expression pattern that should be excluded from both the source and destination. The syntax of the patterns is `regex(5)` (extended syntax). The items actually compared to the pattern are paths relative to the `remote_directory`. The paths always begin with `./`. Any number of `exclude_pattern` options can be specified. For example:

```
exclude_pattern="internal"
```

would exclude all subdirectory paths containing the name `internal`.

Renaming and Deleting Files During Deployment

These options apply to `local_directory` sections, although they can be specified at a higher level to apply to multiple sections.

All of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`do_deletes`

This specifies that files or directories not existing in the source will be deleted on the destination server. By default they are not.

`rename_suffix=suffix`

This specifies a suffix that will be used to rename files normally deleted on the destination server. For example:

```
rename_suffix=.old
```

will rename all files that would otherwise have been deleted to *filename.old*.

Changing Permissions on Files During Deployment

UNIX destination

These local directory options allow you to specify changes in permissions when you are deploying to a UNIX server. Options marked with an asterisk (*) can only be used when you are deploying from a UNIX server to a UNIX server. All of these options apply to *local_directory* sections, although they may be specified at a higher level to apply to multiple sections.

All of these options except for *group_translations* and *user_translations* can also be specified in the server configuration file. Their behavior will depend on how the *client_is_trusted* option is specified in the server configuration file. For more information, see page 290.

```
amask=mask
```

This specifies a bit mask (in octal) to be ANDed with the permission bits of all files and directories. If you are deploying from a Windows NT/2000 server, the bit mask will be ANDed with the default permission bits of 664. For example:

```
amask=011
```

```
dir_perm=permission
```

This specifies the permissions (in octal) given to all deployed directories. For example:

```
dir_perm=755
```

```
file_perm=permission
```

This specifies the permissions (in octal) given to all deployed files. For example:

```
file_perm=755
```

`group=groupid`

This specifies the group assigned to all deployed files and directories. *groupid* must be a valid group name. For example:

```
group=TeamSite_users
```

If `group` is specified, `user` must also be specified.

* `group_translations`

This begins a `group_translations` section that looks like this:

```
group_translations
    1=2
    2=3
    3=4
;
```

Each line specifies a source gid and an equivalent destination gid. Source file and directory gids are translated on transfer to the destination server.

`ignore_groups`

This specifies that changes in file and directory group ownership are ignored when comparing source and destination. By default changes in group ownership are grounds for transfer.

`ignore_modes`

This specifies that changes in file and directory permissions are ignored when comparing source and destination. By default changes in permissions are grounds for transfer.

`ignore_users`

This specifies that changes in file and directory ownership are ignored when comparing source and destination. By default changes in ownership are grounds for transfer.

`omask=mask`

This specifies a bit mask (in octal) to be ORed with the permission bits of all files and directories. If you are deploying from a Windows NT/2000 server, the bit mask will be ORed with the default permission bits of 664. For example:

```
omask=011
```

`user=userid`

This specifies the user who will own all deployed files and directories. *userid* must be a valid user name. For example:

```
user=root
```

If `user` is specified, `group` must also be specified.

* `user_translations`

This begins a `user_translations` section that looks like this:

```
user_translations
    1=2
    2=3
    3=4
;
```

Each line specifies a source uid and an equivalent destination uid. Source file and directory uids are translated on transfer to the destination server.

Windows NT/2000 destination

These local directory options allow you to specify changes in permissions when you are deploying to a Windows NT/2000 server. By default, files will inherit permissions from their parent directories.

All of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290. Note that to use these options you must run OpenDeploy as a user who has “Act as part of the operating system” privileges.

`setaccess=ACL`

Replaces the access control lists (ACLs) for the deployed files and directories.

`changeaccess=ACL`

Modifies the ACL so that the specified users have the specified rights. The new access control entry (ACE) for each specified user allows only the specified rights, discarding any existing ACE.

Windows NT/2000 ACLs

ACLs on Windows NT/2000 have the following syntax (where ACE stands for access control entry):

```
name:ACE  
{ name:ACE, name:ACE, ... }
```

name is one of

```
user name  
group name  
domain name\user name  
domain name\group name
```

ACEs consist of either *perm bits* or *standard perms*.

perm bits is any sequence made of the characters R (read), W (write), X (execute), D (delete), P (change permissions), and O (take ownership), e.g. RWX.

standard perms is one of the following:

```
ALL (RWXDPO)  
NONE (none)  
READ (RX)  
WRITE (W)  
CHANGE (RWXD)
```

For example:

```
setaccess={ andre:ALL, everyone:RX }
```

would remove the existing ACL and grant the user `andre` full access and the group `everyone` read access to the specified files.

```
changeaccess={ chris:ALL, everyone:RX }
```

would remove any existing ACEs for `chris` and `everyone`, and grant `chris` full access and the group `everyone` read access to the specified files. Any other existing ACEs would remain unchanged.

Encryption

OpenDeploy allows two types of encryption: key file and SSL. These types of encryption cannot be used in conjunction with one another, that is, if you use the `key_file` option, you cannot use the SSL options, and vice versa. For more information on OpenDeploy and encryption, see “Encryption” on page 305.

`key_file` is a deployment option, so you can specify different key files for different deployments, or you can specify it at the global level and use one key file for all deployments.

The SSL options are global options. These options will apply to all deployments in a given configuration file. Before you use SSL, you must generate certificates and keys on both the source and destination servers.

`key_file=path`

This specifies the path to the file that will be used as an encryption key for transfers between the `iwdeploy` client and the `iwdeploy` server.

`ssl_certificate=path`

(Mandatory for SSL encryption) This specifies the path to the SSL public key certificate.

`ssl_privatekey=path`

(Mandatory for SSL encryption) This specifies the path to the SSL private key.

`ssl_ciphers=ciphers`

(Optional for SSL encryption) This specifies the SSL ciphers to use. Multiple ciphers must be separated by a colon (:), e.g.:

```
ssl_ciphers=EDH-DSS-DES-CBC3-SHA:EXP-EDH-DSS-DES-CBC-SHA
```

Deploy and Run

OpenDeploy's Deploy and Run feature allows you to specify external scripts to run at various stages of deployment. For more information on Deploy and Run, see "Deploy and Run" on page 313.

Deploy and Run requires you to create a `deploy_run_script` section within a deployment section of the configuration file. Deploy and Run options cannot be specified at a higher level. All of these options can also be specified in the server configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`deploy_run_script=script_to_run`

(Mandatory) This begins a new `deploy_run_script` section. `script_to_run` must be in the current PATH (e.g. `/usr/local/bin`), and the line can contain parameters. For example:

```
deploy_run_script=joe_bob -r -q foobar
```

`as=username`

(UNIX only, optional) This option is specified in a `deploy_run_script` section, and allows you to run the script as a different user. By default, the script runs as the user who invokes OpenDeploy, who will need to be root for most purposes.

`when=condition`

(Mandatory) This option is specified in a `deploy_run_script` section, and allows you to specify when the script is to be run. `condition` is one of:

<code>client_before_deploy</code>	Execute the script on the client, before deployment.
<code>client_after_deploy</code>	Execute the script on the client, after deployment.
<code>server_before_deploy</code>	Execute the script on the server, before deployment.
<code>server_after_deploy</code>	Execute the script on the server, after deployment.
<code>server_before_file</code>	Execute the script on the server, before an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.

<code>server_after_file</code>	Execute the script on the server, after an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.
<code>server_before_dir</code>	Execute the script on the server, before a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.
<code>server_after_dir</code>	Execute the script on the server, after a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.

`client_after_deploy`, `server_after_deploy`, `server_after_file`, and `server_after_dir` may use one of two modifiers, `on_success` or `on_failure`. For example:

```
when=client_after_deploy on_failure
```

would specify an action to be performed on the client after a failed deployment.

If the deployment is a reverse deployment, all scripts will execute on the client.

```
dir_mask=dir
```

This option is specified in a `deploy_run_script` section. `dir` is a regular expression specifying the directories on which the script will be executed, e. g., `*/cgi-bin/*`. The expression matches server-side absolute paths. This option only applies to the `before_dir` and `after_dir` conditions.

```
file_mask=file
```

This option is specified in a `deploy_run_script` section. `file` is a regular expression specifying the files on which the script will be executed, e. g., `.*\.html`. The expression matches server-side absolute paths. This option only applies to the `before_file` and `after_file` conditions.

```
async=yes
```

This option is specified in a `deploy_run_script` section, and will run the script asynchronously. Exercise caution when using this mode, as it could cause many scripts to be run at the same time. The output from scripts run asynchronously is not captured.

`where=dir`

This option is specified in a `deploy_run_script` section. `dir` specifies the directory to navigate to before executing the script.

Links

The options that configure OpenDeploy's behavior with regard to symbolic links do not apply to OpenDeploy for Windows NT/2000.

These options apply to `local_directory` sections, although they may be specified at a higher level to apply to multiple sections. They allow you to specify whether symbolic links should be transferred as-is, or whether they should actually be followed, so that items that they point to are transferred.

`destination_follow_links`

This specifies that symbolic links on the destination server will be followed, i.e., not treated as links, which is the default behavior.

`follow_links`

This specifies that symbolic links on both the source and destination servers will be followed, i.e., not treated as links, which is the default behavior.

`source_follow_links`

This specifies that symbolic links on the source server will be followed, i.e., not treated as links, which is the default behavior.

Debugging Deployment Configuration

OpenDeploy provides an option to facilitate testing of deployment configuration. This option applies to `local_directory` sections, although it may be specified at a higher level in order to apply to multiple sections.

`dont_do`

This specifies that no files should be transferred. The deployment will proceed normally but nothing will be changed on the destination side. This option is commonly used to test changes to deployment configurations.

OpenDeploy Server Options

The OpenDeploy server configuration file allows you to specify:

- Connection options (see page 289)
- Deployment sections (see page 290)
- Security options (see page 290)
- Deployment timeouts (see page 291)
- Encryption options (see page 296)
- Which files to exclude (see page 291)
- Which files to rename or delete during deployment (see page 292)
- Changes to file permissions during deployment (see page 293)
- Deploy and Run options (see page 297)
- Authentication by IP Address (see page 300)

Note that many configuration options can be specified in either the client or the server configuration files. The behavior of these options depends on how the `client_is_trusted` option is specified on the server. If the client is trusted, then the server configuration options are used only if they do not contradict the options specified on the client.

If the client is not trusted, then the server-side options override all options specified on the client, except for specifications that are processed on the client side only.

Specifying Connections and Locations

`port=#`

This global option specifies the port that the `iwdeploy` server will listen to. This must match the port number specified in the client configuration file. This option must be specified.

`TeamSite_server=name`

This global option starts a new `TeamSite_server` section. `name` must be the same as the hostname specified in the OpenDeploy client configuration file. Each server configuration file must include at least one `TeamSite_server` section.

`allowed_directory=path`

This TeamXpress server option specifies an absolute path to a directory into which files can be placed. The directory specified and any of its children are made valid targets. Each `TeamSite_server` section must include at least one `allowed_directory` option. You can specify multiple `allowed_directory` options in a `TeamSite_server` section.

You can also use the `allowed_directory` option as a deployment-level option to specify a section of the server configuration file in the same way that `local_directory` specifies a section of the client configuration file (see the example on page 252).

Specifying Deployment Sections

A single server configuration file can be used to configure several different types of deployment. Each different type of deployment can be independently configured in deployment sections. Each configuration file must have at least one deployment section if `client_is_trusted=no` is specified. To specify a new deployment section, use the `deployment=name` option.

`deployment=name`

This is a `TeamSite_server` option, which begins a deployment section named `name`. If `client_is_trusted=no` is specified, the name of the section must match a deployment section in the client configuration file.

Security Options

`client_is_trusted=yes|no`

This option specifies the behavior of all options that can be specified in both client and server configuration files.

If `client_is_trusted=yes`, then the client configuration file options are used. The server configuration options (that can be specified on the client side) are used only if they do not contradict the options specified on the client. For example, if the client configuration file has a `source_exclude` option and the server configuration file has a `destination_exclude` option, only the `source_exclude` option is used. Or, if the server configuration file has a `deploy_run_script` section for a particular deployment, and the client configuration file does not have a `deploy_run_script` section for that deployment, the server's `deploy_run_script` will not be used.

If `client_is_trusted=no`, then the server-side options override all options specified on the client. The only client options that get used are the name of the deployment section to deploy to, which content on the client-side to deploy, and any other specifications that are processed on the client side only. The `remote_directory` client option will be used if it falls under an `allowed_directory` (as specified on the server).

`client_is_trusted` can be specified at any level. As always, specification of this option at a lower level will supercede specifications at a higher level, for the scope of the lower-level specification only (see “Scope of Configuration File Options” on page 255).

Specifying Deployment Timeouts

```
timeout=#seconds
```

This option specifies the number of seconds it will take for the OpenDeploy server process to time out. `timeout` is a deployment option, but it can also be specified at a higher level. If this option is used, it must be specified on both the client and server.

By default, the server process will time out at 150 seconds. However, if you are doing a TeamXpress comparison-based deployment, you may need to specify a larger number so that the server does not time out before the comparison is completed. For example:

```
timeout=25000
```

Specifying Which Files to Exclude

These options allow you to specify particular files or directories not to deploy. All of these options apply to `allowed_directory` sections, although they can be specified at a higher level to apply to multiple sections. On the OpenDeploy server, you can specify which files to exclude on the destination server only.

Note that although you can specify `exclude` and `exclude_pattern` on the OpenDeploy server, they will behave exactly the same as `destination_exclude` and `destination_exclude_pattern`, respectively. If you specify `source_exclude` or `source_exclude_pattern`, it will be ignored. For more information about excluding files, see page 275.

Excluding Files on the Destination Server

If a file or directory is excluded only on the destination side, and a corresponding file or directory is deployed from the source, the existing file or directory on the destination side will be overwritten. For more information, see page 277.

To exclude files on the destination server, use the `destination_exclude` and `destination_exclude_pattern` options. All of these options can also be specified in the client configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`destination_exclude=path`

This specifies a path *relative* to the `allowed_directory` path which should be excluded from deployment. The effect of this is as if the specified path did not exist on the destination server. If there is a corresponding path on the source side, the destination side will be overwritten (if `client_is_trusted=no`). Any number of `destination_exclude` options can be specified.

`destination_exclude_pattern=pattern`

This specifies a regular expression pattern to exclude on the destination server. The syntax of the patterns is `regex(5)` (extended syntax). The items actually compared to the pattern are paths relative to the `remote_directory`. The paths always begin with `./`. Any number of `destination_exclude_pattern` options can be specified. For example:

```
destination_exclude_pattern="^/htdocs/company/. *html "
```

would exclude all items under subdirectory `/htdocs/company` that end in `html`.

Renaming and Deleting Files During Deployment

These options apply to `allowed_directory` sections, although they can be specified at a higher level to apply to multiple sections.

All of these options can also be specified in the client configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`do_deletes`

This specifies that files or directories not existing in the source will be deleted on the destination server. By default they are not.

`rename_suffix=suffix`

This specifies a suffix that will be used to rename files normally deleted on the destination server. For example:

```
rename_suffix=.old
```

will rename all files that would otherwise have been deleted to *filename.old*.

Changing Permissions on Files During Deployment

UNIX destination

These `allowed_directory` options allow you to specify changes in permissions when you are deploying to a UNIX server. Options marked with an asterisk (*) can only be used when you are deploying from a UNIX server to a UNIX server. All of these options apply to `allowed_directory` sections, although they may be specified at a higher level to apply to multiple sections.

All of these options can also be specified in the client configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`amask=mask`

This specifies a bit mask (in octal) to be ANDed with the permission bits of all files and directories. If you are deploying from a Windows NT/2000 server, the bit mask will be ANDed with the default permission bits of 664. For example:

```
amask=011
```

`dir_perm=permission`

This specifies the permissions (in octal) given to all deployed directories. For example:

```
dir_perm=755
```

`file_perm=permission`

This specifies the permissions (in octal) given to all deployed files. For example:

```
file_perm=755
```

`group=groupid`

This specifies the group assigned to all deployed files and directories. *groupid* must be a valid group name. For example:

```
group=TeamSite_users
```

If `group` is specified, `user` must also be specified.

`ignore_groups`

This specifies that changes in file and directory group ownership are ignored when comparing source and destination systems. By default changes in group ownership are grounds for transfer.

`ignore_modes`

This specifies that changes in file and directory permissions are ignored when comparing source and destination systems. By default changes in permissions are grounds for transfer.

`ignore_users`

This specifies that changes in file and directory ownership are ignored when comparing source and destination systems. By default changes in ownership are grounds for transfer.

`omask=mask`

This specifies a bit mask (in octal) to be ORed with the permission bits of all files and directories. If you are deploying from a Windows NT/2000 server, the bit mask will be ORed with the default permission bits of 664. For example:

```
omask=011
```

`user=userid`

This specifies the user who will own all deployed files and directories. *userid* must be a valid user name. For example:

```
user=root
```

If `user` is specified, `group` must also be specified.

Windows NT/2000 destination

These `allowed_directory` options allow you to specify changes in permissions when you are deploying to a Windows NT/2000 server. By default, files will inherit permissions from their parent directories.

All of these options can also be specified in the client configuration file. Their behavior will depend on how the `client_is_trusted` option is specified in the server configuration file. For more information, see page 290.

`setaccess=ACL`

Replaces the access control lists (ACLs) for the deployed files and directories.

`changeaccess=ACL`

Modifies the ACL so that the specified users have the specified rights. The new access control entry (ACE) for each specified user allows only the specified rights, discarding any existing ACE.

Windows NT/2000 ACLs

ACLs on Windows NT/2000 have the following syntax (where ACE stands for access control entry):

name : ACE

{ *name* : ACE, *name* : ACE, ... }

name is one of

user name

group name

domain name \ *user name*

domain name \ *group name*

ACEs consist of either *perm bits* or *standard perms*.

perm bits is any sequence made of the characters R (read), W (write), X (execute), D (delete), P (change permissions), and O (take ownership), e.g. RWX.

standard perms is one of the following:

ALL (RWXDPO)

NONE (none)

READ (RX)

WRITE (W)

CHANGE (RWXD)

For example:

```
setaccess={ andre:ALL, everyone:RX }
```

would remove the existing ACL and grant the user `andre` full access and the group `everyone` read access to the specified files.

```
changeaccess={ chris:ALL, everyone:RX }
```

would remove any existing ACEs for `chris` and `everyone`, and grant `chris` full access and the group `everyone` read access to the specified files. Any other existing ACEs would remain unchanged.

Encryption

OpenDeploy allows two types of encryption: key file and SSL. These types of encryption cannot be used in conjunction with one another; that is, if you use the `key_file` option, you cannot use the SSL options, and vice versa. For more information on encryption, see “Encryption” on page 305.

`key_file` is a `TeamSite_server`-level option, so you can specify different key files for different servers, or you can specify it at the global level and use one key file for all servers.

The SSL options are global options. These options will apply to all servers in a given configuration file. Before you use SSL, you must generate certificates and keys on both the source and destination servers.

```
key_file=path
```

This specifies the path to the file which will be used as an encryption key for transfers between the `iwdeploy` client and the `iwdeploy` server.

`ssl_certificate=path`

(Mandatory for SSL encryption) This specifies the path to the SSL certificate.

`ssl_privatekey=path`

(Mandatory for SSL encryption) This specifies the path to the SSL private key.

`ssl_ciphers=ciphers`

(Optional for SSL encryption) This specifies the SSL ciphers to use. Multiple ciphers must be separated by a colon (:), e.g.:

```
ssl_ciphers=EDH-DSS-DES-CBC3-SHA:EXP-EDH-DSS-DES-CBC-SHA
```

Deploy and Run

If the client is not trusted, Deploy and Run requires you to create a `deploy_run_script` section within a `deployment` section of the server configuration file. Options that belong in this section cannot be specified at a higher level.

In addition to the `deploy_run_script` options that can be specified on either client or server, Deploy and Run has two server-only options which allow you to specify security-related settings for Deploy and Run scripts (`disable_scripts` and `require_abs_script_path`). These options are specified at the global level. For more information on Deploy and Run, see “Deploy and Run” on page 313.

`disable_scripts=yes`

This specifies that Deploy and Run scripts will be disabled on the server. This global option can only be specified on the server.

`require_abs_script_path=yes`

Requires that all scripts be specified using absolute paths, not relative paths. Scripts specified using relative paths will not be allowed to execute, but the deployment will otherwise proceed normally. This option must be specified at the global level of the OpenDeploy server configuration file. This option can only be specified on the server.

`deploy_run_script=script_to_run`

(Mandatory if `client_is_trusted=no`) This begins a new `deploy_run_script` section. `script_to_run` must be in the current PATH (e.g. `/usr/local/bin`), and the line can contain parameters. For example:

```
deploy_run_script=joe_bob -r -q foobar
```

This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

`as=username`

(UNIX only, optional) This option is specified in a `deploy_run_script` section, and allows you to run the script as a different user. By default, the script runs as the user who invokes OpenDeploy, who will need to be root for most purposes. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

`when=condition`

(Mandatory if `client_is_trusted=no`) This option is specified in a `deploy_run_script` section, and allows you to specify when the script is to be run. `condition` is one of:

<code>server_before_deploy</code>	Execute the script on the server, before deployment.
<code>server_after_deploy</code>	Execute the script on the server, after deployment.
<code>server_before_file</code>	Execute the script on the server, before an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.
<code>server_after_file</code>	Execute the script on the server, after an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.

<code>server_before_dir</code>	Execute the script on the server, before a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.
<code>server_after_dir</code>	Execute the script on the server, after a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.

`server_after_deploy`, `server_after_file`, and `server_after_dir` may use one of two modifiers, `on_success` or `on_failure`. For example:

```
when=server_after_deploy on_failure
```

would specify an action to be performed on the server after a failed deployment.

If the deployment is a reverse deployment, all scripts will execute on the client. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

```
dir_mask=dir
```

This option is specified in a `deploy_run_script` section. *dir* is a regular expression specifying the directories on which the script will be executed, e. g., `*/cgi-bin/*`. The expression matches server-side absolute paths. This option only applies to the `before_dir` and `after_dir` conditions. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

```
file_mask=file
```

This option is specified in a `deploy_run_script` section. *file* is a regular expression specifying the files on which the script will be executed, e. g., `.*\.html`. The expression matches server-side absolute paths. This option only applies to the `before_file` and `after_file` conditions. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

```
async=yes
```

This option is specified in a `deploy_run_script` section, and will run the script asynchronously. Exercise caution when using this mode, as it could cause many scripts to be run at the same time. The output from scripts run asynchronously is not captured. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

where=dir

This option is specified in a `deploy_run_script` section. *dir* specifies the directory to navigate to before executing the script. This option can be specified on either the client or server. Its behavior is dependent on the `client_is_trusted` option (see page 290).

Authentication by IP Address

OpenDeploy has two server options that can work with your firewall to ensure that the OpenDeploy listener is communicating with a known server in a known manner. For more information about authentication by IP address, see “Authentication by IP Address” on page 301.

bind_address=address

where *address* specifies the IP address that the OpenDeploy server will use. The value can be a hostname, which will be validated by a DNS lookup via `gethostbyname()`, or an IP address, which will be validated by a check via `inet_addr()`. When `bind_address` is specified together with `port`, OpenDeploy will `bind()` on the specified `bind_address` and `port`. If `bind_address` is not specified, OpenDeploy will `bind()` only on the specified port and listen on all interfaces.

`bind_address` is a global section option.

allowed_hosts=hostlist

hostlist is a list of the OpenDeploy senders which will be allowed to connect to the OpenDeploy listener. The list can be space-delimited or comma-delimited, and you can specify either hostnames or dot-notations. `localhost` and `127.0.0.1` are not valid values within this list. When an incoming connect request is received, the incoming connecting IP address will be matched against the IP address(es) converted from the `allowed_hosts` list. A match with any address in the list will validate the incoming connection; otherwise the connection will be rejected. `allowed_hosts` is a `TeamSite_server` section option.

Chapter 16

Advanced Features

This chapter discusses several OpenDeploy features that may require configuration in areas outside of the OpenDeploy client and server configuration files. OpenDeploy features that are solely invoked on the command line or that are configured via configuration files are discussed elsewhere in this manual.

This chapter discusses:

- Authentication by IP address (see page 301)
- Encryption (see page 305)
- Deploy and Run (see page 313)

Authentication by IP Address

OpenDeploy can be configured to work with your firewall to ensure that the OpenDeploy listener is communicating with a known server in a known manner.

In the following scenario, OpenDeploy is installed on both a single development server and a single production server. On the production server, OpenDeploy is installed as a service (Windows NT/2000) or a daemon (UNIX). There is a firewall between the development and production servers, with all outbound traffic connecting first to the firewall and then to the external location. The external production server is configured with two IP addresses, only one of which is publicly visible for accepting web traffic.

Internal network: 10.1.1.0

External network: 10.2.2.0

On internal network:

Web source system: 10.1.1.1

Firewall: 10.1.1.2

On external network:

Firewall: 10.2.2.2

Web server:

Private: 10.2.2.3

Public: 10.2.2.4

On the production server, OpenDeploy listens on 10.2.2.3 at port 1701. OpenDeploy communicates securely through the firewall to the OpenDeploy client as follows:

10.1.1.1:(*random1*) ⇒ 10.1.1.2:50001 [proxy]

10.2.2.2:(*random2*) ⇒ 10.2.2.3:1701

Note: *random1* and *random2* are any dynamically assigned source port numbers.

An ⇒ shows a TCP connection from source to destination.

Source and destination are *address:port*.

[proxy] is the process on the firewall that's performing the proxying.

In the above example, the OpenDeploy client on the development server would specify the IP address and port number of the firewall proxy. The firewall proxy, upon making a connection with the OpenDeploy client, would create a separate connection to the development server and the development server would communicate directly with the firewall on the production server.

The OpenDeploy server on the production server can be set to listen for connections on a specific interface. In this example, the OpenDeploy server would only receive connections made by the firewall at 10.2.2.3 on port 1701 and not on any other IP address (such as the public IP address assigned to the webserver).

Furthermore, the OpenDeploy server on the production server can be set to receive content only from a known, trusted source. In this example, the OpenDeploy server can be set to only receive content from a source IP address of 10.2.2.2—the external address of the firewall. With this option set, even if an outside user could make a connection to the right IP address at the right port, that user would need to identify the appropriate internal IP address of the client to establish a connection. The following diagram shows the sequence of events that occurs when the client tries to connect to the server:

Client		Server	
⇐	Connect	⇒	
⇐	Connect-ack	⇒	
⇐	IP Checking	⇒	Server checks that client is an allowed host.
⇐	Session-info	⇒	
⇐	Difference-info	⇒	Server issues “get” requests to client.
⇐	Content-transfer	⇒	Files are sent over with ack/nack returned
⇐	End-session-info	⇒	

High-level Protocol Diagram

Authentication by IP address uses the following options in the OpenDeploy server configuration file. These options may be used together or separately:

`bind_address=address`

where *address* specifies the IP address that the OpenDeploy server will use. The value can be a hostname, which will be validated by a DNS lookup via `gethostbyname()`, or an IP address, which will be validated by a check via `inet_addr()`. When `bind_address` is specified together with `port`, OpenDeploy will `bind()` on the specified `bind_address` and `port`. If `bind_address` is not specified, OpenDeploy will `bind()` only on the specified port and listen on all interfaces.

`bind_address` is a global section option.

`allowed_hosts=hostlist`

`hostlist` is a list of the OpenDeploy senders that will be allowed to connect to the OpenDeploy listener. The list can be space-delimited or comma-delimited, and you can specify either hostnames or dot-notations. `localhost` and `127.0.0.1` are not valid values within this list. When an incoming connect request is received, the incoming connecting IP address will be matched against the IP address(es) converted from the `allowed_hosts` list. A match with any address in the list will validate the incoming connection; otherwise the connection will be rejected. `allowed_hosts` is a `TeamSite_server` section option.

If the firewall between the sender and receiver is configured for packet filtering, the OpenDeploy client `remote_server`, `remote_port`, and `hostname` must match the server's `bind_address`, `port`, and `TeamSite_server`, respectively. If the firewall is configured for proxy operation, the server's `allowed_hosts` must include the external firewall interface.

The following OpenDeploy server configuration file shows how the `bind_address` and `allowed_host` options are used.

Example

```
#
# SERVER-SIDE OPENDEPLOY
#
port=1709
bind_address=204.247.119.36
TeamSite_server=TeamSite1.example.com
  allowed_hosts=204.247.119.36
  key_file=/u/iw/andre/deploytest/encryptkey
  allowed_directory = /tmp/unixunix/deploydst
  allowed_directory = /tmp/Branch1
;
```

If the `bind_address` check fails, the following error message will appear:

```
ERROR: Current host not allowed to run iwdeploy as daemon
```

If the `allowed_hosts` check fails, the following error message will appear:

```
ERROR: Connecting host denied access
```

Encryption

OpenDeploy provides two methods of encryption: weak (40-bit) symmetric and strong (up to 168-bit) asymmetric key encryption.

Symmetric Key Encryption

OpenDeploy's `key_file` option uses a symmetric key algorithm to provide 40-bit encryption support for content transfers. To configure OpenDeploy for symmetric key deployment, the same encryption key file must exist on both the source and the destination server. OpenDeploy's symmetric key deployment provides basic encryption support with minimal performance impact on content deployment. However, symmetric 40-bit encryption is breakable by brute force attack with a modest amount of computing power and is potentially vulnerable to unauthorized users with the same symmetric key who can intercept data passing over the wire. For sites requiring stronger guarantees against brute force attacks, and for sites requiring complete certainty that data is only being received or transmitted by a trusted source, OpenDeploy provides 168-bit asymmetric encryption (discussed later in this chapter). The following section describes how to specify key files to implement 40-bit symmetric key encryption.

Key Files

To specify key file (symmetric) encryption, add the following line to both the client and server configuration files:

```
key_file=path-to-keyfile
```

This specifies the path to the file used as an encryption key for transfers between the `iwdeploy` client and the `iwdeploy` server. It must be specified as a deployment option in the client configuration file and a TeamXpress server option in the server configuration file.

You can use any file as a key file, but you must use the same file on both the development and the production servers. For cross platform deployments, key files that are plain text files must match exactly. When using FTP to transfer key files, be sure to use binary mode.

Asymmetric Key Encryption

OpenDeploy provides up to 168-bit asymmetric key encryption support for secure content transfers. This deployment option uses a certificate authority (provided with OpenDeploy) to ensure that all content transfers occur only between known, trusted sources. OpenDeploy's asymmetric key algorithm uses public key exchange to authenticate destination servers, and can use any one of a number of algorithms to encrypt content prior to deployment. OpenDeploy's asymmetric key deployment offers the strongest possible encryption support and is unlikely to be broken by any brute force attack. Furthermore, when used in conjunction with digital certificates for initial server authentication, OpenDeploy's asymmetric key encryption is not vulnerable to a man-in-the-middle attack. OpenDeploy's asymmetric key encryption provides the strongest possible security support in exchange for a small performance cost.

168-bit encryption is available only for transfers within the United States of America. For more information about encryption and ciphers, consult a cryptography reference manual such as *Applied Cryptography* (Bruce Schneier, ISBN 0-471-11709-9). You can also set up asymmetric encryption to provide less than full 168-bit security. See the section "Configuring OpenDeploy for Asymmetric Encryption" later in this chapter for information about using various ciphers to set different levels of encryption.

Setting Up SSL

For asymmetric encryption, OpenDeploy uses the SSLey implementation of SSL.

Before you use OpenDeploy with asymmetric encryption, you must perform the following steps. These steps create two unique public and private key pairs that are signed by the same certificate authority. One key pair will be copied to the production server, and be used by the server component of OpenDeploy. The other key pair will be copied to the development server, and be used by the client component of OpenDeploy. You must perform all of these steps no matter what level of asymmetric encryption you intend to set up.

The certificate authority consists of a set of programs used for generating public and private key pairs and a database containing state information. The programs will be installed in `opendeploy/bin`. The database, by default, is contained in the directory where the programs are run. If future public and private key pairs are created using a different certificate authority, OpenDeploy will not be able to deploy to or from a host with keys created by an older certificate authority.

Pass Phrases

In the following steps, you will be prompted as follows for a pass phrase:

```
Enter PEM pass phrase:
```

Give the following response whenever you are prompted:

```
1234
```

OpenDeploy requires this particular pass phrase to use the generated certificates.

Setting up the Certificate Authority

UNIX

1. Create the directory where the certificate authority will be installed, e.g., `opendeploy/conf/ca`. Navigate to that directory.
2. Verify that the `opendeploy/bin` directory is included in the `PATH` environment variable.
3. Copy the `ssleay.cnf` file from the `opendeploy/bin` directory into the current working directory.
4. Install the new certificate authority:

```
% CA.sh -newca
```

When prompted for a pass phrase, enter 1234.

Windows NT/2000

To set up the SSLeay DSA certificate authority:

1. Start a command prompt, `cmd.exe`. Verify that the `PATH` environment variable contains `OpenDeploy\bin`:

```
>set PATH
```

If the `PATH` does not contain the `OpenDeploy\bin` directory, add it now:

```
>set PATH=%PATH%;opendeploy\bin
```



2. Create the directory where the certificate authority will be installed, e.g., `OpenDeploy\conf\ca`. Navigate to that directory.
3. Copy the `ssleay.cnf` file from the `OpenDeploy\bin` directory into the current working directory.

4. Generate a DSA parameter file:

```
>ssleay dsaparam 512 -out iwoddsa512.pem
```

5. Generate a DSA certificate (the passphrase is 1234):

```
>ssleay req -config ssleay.cnf -x509 -newkey dsa:iwoddsa512.pem -out iwoddsaca.pem
```

When prompted for a pass phrase, enter 1234.

This step also creates a private key in file `privkey.pem` and a public key in file `iwoddsaca.pem`.

6. Check the newly generated certificate:

```
>ssleay x509 -text -in iwoddsaca.pem
```

7. Build the certificate authority directory structure and supporting files:

```
>mkdir demoCA
>mkdir demoCA\certs
>mkdir demoCA\crl
>mkdir demoCA\private
>mkdir demoCA\newcerts
>echo 01 > demoCA\serial
>copy nul demoCA\index.txt
```

8. Move the files to their correct places:

```
>move privkey.pem demoCA\private\cakey.pem
>move iwoddsaca.pem demoCA\certs\cacert.pem
```

Generating a Certificate

UNIX

To generate a DSA Certificate for OpenDeploy, do the following once for the development and once for the production server:

1. Generate a new certificate and key:

```
% CA.sh -certall
```

When prompted for a pass phrase, enter 1234.

This step generates a private key file called `privkey.pem` and a certificate file called `newdhsigned.pem`.

2. Copy the generated keys to the appropriate locations, depending on whether the certificate/key pair is intended for the client or server component of OpenDeploy. A good place to store certificates and keys is `opendeploy/cert`. This directory is not created by the installation process; you will have to create it manually. You might also want to rename the keys to reflect their role in the deployment cycle, e.g. client side keyfiles may be called `odcltkey.pem` and `odcltcert.pem`, while server side keyfiles may be called `odsvrkey.pem` and `odsvrcert.pem`.

Windows NT/2000

To generate a DSA Certificate for OpenDeploy, do the following once for the development and once for the production server:

1. Generate the DSA Certificate:

```
>ssleay req -config ssleay.cnf -newkey dsa:iwoddsa512.pem -out  
yournewreq.pem
```

When prompted for a pass phrase, enter 1234. However, the challenge password can be any value.

This step generates a private key in file `privkey.pem`, which should be renamed:

```
>move privkey.pem odkey.pem
```

2. Sign the certificate:

```
>ssleay ca -config ssleay.cnf -in yournewreq.pem -out odcert.pem
```

When prompted for a pass phrase, enter 1234.

3. Generate Diffie-Hellman parameters and append them to the certificate:

```
>ssleay gendh -rand odcert.pem -out dh.out  
>type dh.out >> odcert.pem
```

4. Copy the generated keys to the appropriate locations, depending on whether the certificate/key pair is intended for the client or server component of OpenDeploy. A good place to store certificates and keys is `OpenDeploy/cert`. This directory is not created by the installation process; you will have to create it manually. You might also want to rename the keys to reflect their role in the deployment cycle, e.g. client-side keyfiles could be called `odcltkey.pem` and `odcltcert.pem`, while server-side keyfiles could be called `odsvrkey.pem` and `odsvrcert.pem`.

Configuring OpenDeploy for Asymmetric Encryption

After generating and signing the certificates as described in the preceding sections, you must configure OpenDeploy to use asymmetric encryption.

Configuration Options

To configure OpenDeploy to use SSL, specify the generated certificate and key files in the `iwdeploy` client and server configuration files as global options:

```
ssl_certificate=path  
ssl_privatekey=path
```

You can also specify various ciphers to use in encryption. During a connection, the OpenDeploy client and server will negotiate which cipher to use. During the negotiation phase, OpenDeploy selects the highest priority cipher that both client and server support. Specify ciphers as follows:

```
ssl_ciphers=cipherlist, where
```

cipherlist contains one or more ciphers, ranked left to right from highest priority to lowest priority, separated by a colon (:), e.g.:

```
ssl_ciphers=EDH-DSS-DES-CBC3-SHA:EXP-EDH-DSS-DES-CBC-SHA
```

`ssl_ciphers` is a global option, and it can be specified in the OpenDeploy client or server configuration file, or in both configuration files. If `ssl_ciphers` is not specified, the default is:


```
ssl_ciphers=EDH-DSS-DES-CBC3-SHA:EDH-DSS-DES-CBC-SHA:ADH-DES-CBC3-SHA:ADH-DES-CBC-SHA:EXP-ADH-DES-CBC-SHA
```

Currently the only 168-bit cipher available is EDH-DSS-DES-CBC3-SHA.

Supported Ciphers

OpenDeploy allows you to use the following ciphers:

No-authentication ciphers

```
ADH-DES-CBC3-SHA  
ADH-DES-CBC-SHA
```

Low strength ciphers

```
EDH-DSS-DES-CBC-SHA
```

High strength ciphers

```
EDH-DSS-DES-CBC3-SHA
```

Export ciphers

```
EXP-EDH-DSS-DES-CBC-SHA  
EXP-ADH-DES-CBC-SHA
```

Sample Server Configuration Files

UNIX

```
port=1709  
ssl_certificate=/usr/opendeploy/conf/odsvrcert.pem  
ssl_privatekey=/usr/opendeploy/conf/odsvrkey.pem  
ssl_ciphers=EDH-DSS-DES-CBC-SHA:EXP-ADH-DES-CBC-SHA  
TeamSite_server=development1.example.com  
    allowed_directory = /tmp/deploydst  
;
```

Windows NT/2000

```
port=1709  
ssl_certificate=C:\Program Files\Interwoven\OpenDeploy\conf\odsvrcert.pem  
ssl_privatekey=C:\Program Files\Interwoven\OpenDeploy\conf\odsvrkey.pem
```



```
ssl_ciphers=EDH-DSS-DES-CBC-SHA:EXP-ADH-DES-CBC-SHA
TeamSite_server=development1.example.com
    allowed_directory = D:\deploydst\content
;
```

Sample Client Configuration Files

UNIX

```
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709
ssl_certificate=/usr/iw-home/opendeploy/conf/odcltcert.pem
ssl_privatekey=/usr/iw-home/opendeploy/conf/odcltkey.pem
ssl_ciphers=EDH-DSS-DES-CBC-SHA:EXP-ADH-DES-CBC-SHA
deployment=deploy_to_single
    area=/u/iw/andre/deploy
    local_directory=deploysrc
    remote_directory=/tmp/deploydst
;
```

Windows NT/2000

```
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709
ssl_certificate=C:\Program Files\Interwoven\OpenDeploy\conf\odcltcert.pem
ssl_privatekey=C:\Program Files\Interwoven\OpenDeploy\conf\odcltkey.pem
ssl_ciphers=EDH-DSS-DES-CBC-SHA:EXP-ADH-DES-CBC-SHA
deployment=deploy_to_single
    area=y:\default\main\dev\EDITION
    local_directory=deploysrc
    remote_directory=D:\deploydst\content
;
```

Deploy and Run

OpenDeploy’s “Deploy and Run” feature allows you to configure OpenDeploy to execute an external script at a specified stage of deployment. For example, OpenDeploy can be configured to execute a notification script upon a failed deployment, run a language-checking script during deployment, or enter items in a Windows NT/2000 server’s Registry after deployment.

Configuring Deploy and Run

If the client is trusted, most Deploy and Run configuration is done in the client configuration file, although there are also two server configuration file options. The options described in “Client Configuration” may also be specified on the server. Their behavior is dependent on whether or not the client is trusted (see “Use of Client versus Server Configuration Options” on page 258). The following sections describe the changes you can make to both files to set up Deploy and Run.

Client Configuration

To configure Deploy and Run, add a `deploy_run_script` section to a deployment section of an OpenDeploy client configuration file. `deploy_run_script` options must be contained within a deployment section; they cannot be specified at a higher level. However, a single deployment section can contain multiple `deploy_run_script` sections that include different options. The `deploy_run_script` line specifies the script to run and its parameters:

```
deploy_run_script=script_to_run
```

(Required) This line can contain parameters, e.g., `deploy_run_script=myscript -r -q`

script_to_run must be in the current PATH (e.g., `/usr/local/bin`).

A `deploy_run_script` section can contain the following lines:

```
as=username
```

(UNIX only, optional) This option allows you to run the script as a different user. By default, the script runs as the user who invokes OpenDeploy, who will need to be root for most purposes.

```
when=condition
```

(Required) where *condition* is one of the following:

<code>client_before_deploy</code>	Execute the script on the client, before deployment.
<code>client_after_deploy</code>	Execute the script on the client, after deployment.
<code>server_before_deploy</code>	Execute the script on the server, before deployment.
<code>server_after_deploy</code>	Execute the script on the server, after deployment.
<code>server_before_file</code>	Execute the script on the server, before an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.
<code>server_after_file</code>	Execute the script on the server, after an individual file is deployed (may be used in conjunction with the <code>file_mask</code> and <code>dir_mask</code> options). Exercise caution when using this option, as it can slow deployment and cause log files to become extremely large. This option cannot be used for transactional mode deployments.
<code>server_before_dir</code>	Execute the script on the server, before a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.
<code>server_after_dir</code>	Execute the script on the server, after a directory is deployed (may be used in conjunction with the <code>dir_mask</code> option). This option cannot be used for transactional mode deployments.

`client_after_deploy`, `server_after_deploy`, `server_after_file`, and `server_after_dir` may use one of two modifiers, `on_success` or `on_failure`. For example:

```
when=client_after_deploy on_failure
```

would specify an action to be performed on the client after a failed deployment.

If the deployment is a reverse deployment, all scripts will execute on the client.

```
dir_mask=dir
```

(Optional) where *dir* is a regular expression specifying the directories on which the script will be executed, e. g. `*/cgi-bin/*`. The expression matches server-side absolute paths. This option only applies to the `before_dir` and `after_dir` conditions.

`file_mask=file`

(Optional) where *file* is a regular expression specifying the files on which the script will be executed, e. g., `.*\.html`. The expression matches server-side absolute paths. This option only applies to the `before_file` and `after_file` conditions.

`async=yes`

(Optional) This option will run the script asynchronously. Exercise caution when using this mode, as it could cause many scripts to be run at the same time. The output from scripts run asynchronously is not captured.

`where=dir`

(Optional) where *dir* specifies the directory to navigate to before executing the script.

Examples

The following client configuration file contains two `deploy_run_script` sections. The first `deploy_run_section` specifies that the `mail_info` script is to run with the parameter `after_deploy`. It is to run on the client after the deployment is completed, whether it succeeds or fails.

The second `deploy_run_section` specifies that the `check_log_file` script is to be run in the `/home/andre` directory on the target before and after every successful deployment of a directory or of a file whose name ends in `.log`.

```
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709
deployment=manyscripts
  area=/u/iw/andre/deploy
  local_directory=deploysrc
  remote_directory=/tmp/deploydst
;
deploy_run_script=mail_info -after_deploy
  when=client_after_deploy on_success on_failure
  async=yes
;
```



```
deploy_run_script=check_log_file
  when=before_dir after_dir before_file after_file on_success
  where=/home/andre
  file_mask=*.log
;
;
```

The following configuration file invokes a script that sends email to the system administrator when a deployment fails. The script to send mail is included on the OpenDeploy CD-ROM.

```
hostname = development1.example.com
deployment = website
  remote_server = production1.example.com
  remote_port= 1849
  key_file = /local/deploy/key/web_key
  deploy_run_script=/local/deploy/script/mail-admin
    as=andre
    when=server_after_deploy on_failure
  ;
  area>//IWSERVER/default/main/dev/EDITION
  local_directory = .
    remote_directory = /local/docroot
    do_deletes
  ;
;
```

The following configuration file invokes a script that stops and restarts the webserver on the production server before the deployment starts. The production server in this example is a Windows NT/2000 server running Microsoft IIS. The scripts to start and stop the webserver are included on the OpenDeploy CD-ROM.

Due to the constraints of page width, some of the lines in the configuration file below may appear to wrap. Lines in an actual OpenDeploy configuration file should never wrap.

```
hostname = development1.example.com
deployment = website-nt
  remote_server = production1.example.com
  remote_port= 1849
  key_file = d:\deploy\website-nt.key
```

```

    deploy_run_script=
    "d:\program files\interwoven\TeamXpress\iw-perl\bin\iwperl.exe"
    d:\deploy\script\stop-iis.ipl
        when=server_before_deploy
        ;
    deploy_run_script=
    "d:\program files\interwoven\TeamXpress\iw-perl\bin\iwperl.exe"
    d:\deploy\script\start-iis.ipl
        when=server_after_deploy
        ;
    area=//IWSERVER/default/main/dev/EDITION
    local_directory = .
        remote_directory = d:\website
    ;
;

```

Further sample configuration files and their corresponding scripts are available on the OpenDeploy CD-ROM.

Server Configuration

Deploy and Run uses two global server configuration options. Server configuration files can also contain a `deploy_run_script` section which specifies the same options as are used on the client. This section will only be used if the client is not trusted.

`disable_scripts=yes`

Disables the Deploy and Run feature. To disable Deploy and Run, include this option at the global level of the OpenDeploy server configuration file.

`require_abs_script_path=yes|no`

Requires that all scripts be specified using absolute paths, not relative paths. Scripts specified using relative paths will not be allowed to execute, but the deployment will otherwise proceed normally. This option must be specified at the global level of the OpenDeploy server configuration file.

Examples

The following server configuration file disables the Deploy and Run feature:

```

port=1709
disable_scripts=yes

```

```
TeamSite_server=development1.example.com
  key_file=/u/iw/andre/deploy/encryptkey
  allowed_directory = /tmp/deploydst
;
```

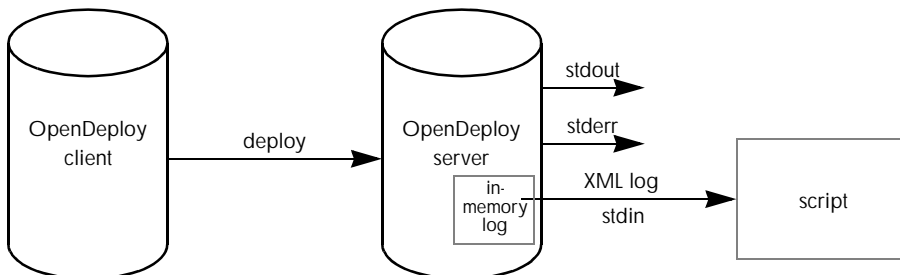
The following server configuration file requires Deploy and Run scripts to use absolute paths:

```
port=1709
require_abs_script_path=yes
TeamSite_server=development1.example.com
  key_file=/u/iw/andre/deploy/encryptkey
  allowed_directory = /tmp/deploydst
;
```

Log Files and Scripts

The following steps execute whenever Deploy and Run calls a script that executes on the OpenDeploy server:

1. The OpenDeploy server spawns a process that will execute the script.
2. If Deploy and Run is set to run scripts synchronously, a pipe is created between the OpenDeploy server process and the script process. The script will receive input through `stdin` from the OpenDeploy server process, and it will write output through `stdout` and `stderr` to the OpenDeploy server process.
3. `stdin` of the script process receives an XML representation of the OpenDeploy log file in its current state.

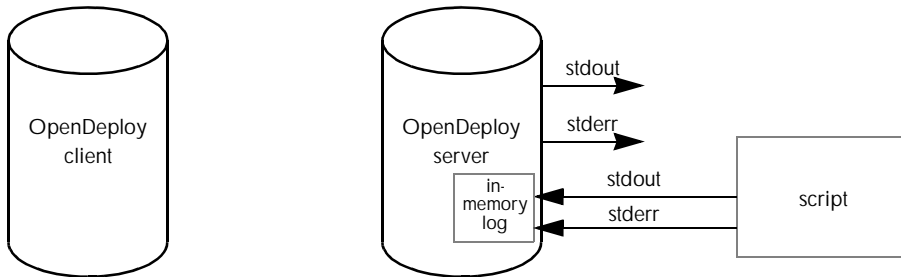


4. The script executes, and the results are sent to `stdout`. Errors are sent to `stderr`.

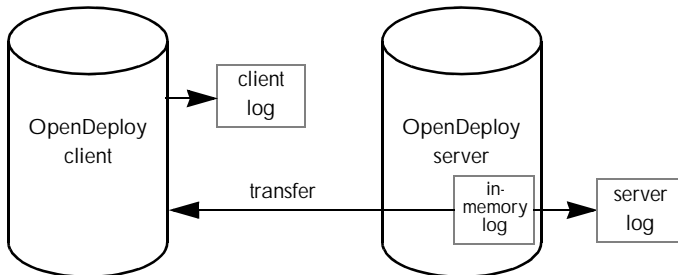
If the scripts are not run in asynchronous mode, the OpenDeploy process receives the results and sends them to the OpenDeploy log (results of scripts run in asynchronous mode are not logged). If the results are well-formed (that is, they conform to the XML DTD), they are parsed and sent to the log as XML objects. If they are not well-formed, they are not parsed, but they are still sent to the log.

Because well-formed results are parsed by the server, it is possible for a script to cause OpenDeploy to abort deployment by generating certain responses in XML, e. g., `result="-2"` (see "The OpenDeploy Log File DTD" on page 320).

This process is repeated each time Deploy and Run calls a script.



- 5. At the end of a deployment, the server log is written to a file, and transferred to the client (see "Logging" on page 243 for an explanation of client and server log files).



On UNIX, output from scripts is appended to the OpenDeploy log file. On Windows NT/2000, however, each deployment session creates a new trace log file, named as described on page 247. Script output is stored in a separate trace log file from the server trace log.



In this manner, future scripts can parse the output of past scripts. For example, a script might extract information about which files were deleted during the last deployment. Sample code illustrating how to parse the OpenDeploy log file is included on the OpenDeploy CD-ROM. Also see “Parsing the OpenDeploy Log File” on page 322.

The OpenDeploy Log File DTD

The XML representation of the log file has the following DTD:

```
<!DOCTYPE log [
  <!ELEMENT log ANY>
  <!ATTLIST log target CDATA "">
  <!ATTLIST log action CDATA "0">
  <!ATTLIST log date CDATA "0">
  <!ATTLIST log result CDATA "0">
  <!ATTLIST log response CDATA "">
  <!ELEMENT log_element ANY>
  <!ATTLIST log_element target CDATA "">
  <!ATTLIST log_element action CDATA "0">
  <!ATTLIST log_element date CDATA "0">
  <!ATTLIST log_element result CDATA "0">
  <!ATTLIST log_element response CDATA "">
]>
```

In an OpenDeploy XML log file:

`target` specifies the target, e.g., the name of the file or directory deployed.

`action` is one of the following numbers:

#	Name	Description
0	UNDEFINED	Undefined action.
1	LOG_ELEMENT_DEPLOY_FILE	Deploy a file.
2	LOG_ELEMENT_DEPLOY_DIRECTORY	Deploy a directory.
3	LOG_ELEMENT_SUMMARY	Contains summary of events.
4	LOG_ELEMENT_RUN_SCRIPT	Run a script.

`date` specifies the date (in number of seconds since January 1, 1970).

`result` is one of the following numbers:

#	Name	Description
-2	LOG_ELEMENT_ABORT	The result of the action was to signal the deployment to do a hard abort (the deployment stops immediately. If the deployment was transactional, the files on the production server will be returned to their original state).
-1	LOG_ELEMENT_ERROR	The action returned an error.
0	LOG_ELEMENT_OK	The action returned a satisfactory result and the item should be printed to <code>stdout</code> in all cases.
1	LOG_ELEMENT_OK1	The action returned a satisfactory result which should be printed to <code>stdout</code> if the verbosity level is set to a value greater than or equal to 1.
2	LOG_ELEMENT_OK2	The action returned a satisfactory result which should be printed to <code>stdout</code> if the verbosity level is set to a value greater than or equal to 2.
3	LOG_ELEMENT_OK3	The action returned a satisfactory result which should be printed to <code>stdout</code> if the verbosity level is set to a value greater than or equal to 3.
4	LOG_ELEMENT_OK4	The action returned a satisfactory result which should be printed to <code>stdout</code> if the verbosity level is set to a value greater than or equal to 4.
5	LOG_ELEMENT_OK5	The action returned a satisfactory result which should be printed to <code>stdout</code> if the verbosity level is set to a value greater than or equal to 5.

`response` is the text response for the action.

For example, a log file might contain the following line:

```
<log_element target="/tmp/Branch1/src/pmt" action="2" date="925263642"
result="0" response="" />
```

indicating that the target was `/tmp/Branch1/src/pmt`, the action was to deploy a directory, the result was satisfactory, and there was no text response for this action.

Parsing the OpenDeploy Log File

To parse OpenDeploy XML log files:

1. Install the `XML::Parser` module (packaged with OpenDeploy in the `examples/xml-perl` directory as `xml.tar.gz`, or see your local CPAN mirror <http://www.perl.com/CPAN>). This package requires a version of Perl more recent than 5.004. See the README file for installation directions.
2. Set the search path for modules. Perl looks in well-known places for Perl modules, stored in the `@INC` list. For example, to ensure that the OpenDeploy XML module `IWXML.pm`, in `/usr/opendeploy/lib`, is found by Perl, use the statement

```
@INC = (@INC, "/usr/opendeploy/lib");
```

This line should be followed by:

```
require IWXML;
```

which includes all elements contained within the OpenDeploy XML module into the custom script.

3. Scripts triggered by OpenDeploy 4.2 read input from standard input, process the data, and report back results on standard output. The programming model reflects this. The data will be parsed by the following line of code:

```
my($xml_obj) = IWXML::GetLogDataFromSTDIN();
```

The script can obtain a list of files and directories for which deployment succeeded:

```
my(@f_success) = $xml_obj -> GetSucceededFiles();
```

and a list of files and directories that failed to be deployed:

```
my(@f_failure) = $xml_obj -> GetFailedFiles();
```

The script can process these lists with the necessary logic.

4. To report results back to the OpenDeploy process, print a single line of XML to standard output:

```
printf (<response code=\"%d\">\n\", $retval);
```

where `$retval` corresponds to one of the result values listed on page 321. Note especially that a value of `-2` will cause the deployment to be aborted.



Chapter 17

Deployment Scenarios

This chapter describes the most common deployment scenarios and provides examples of the configuration files necessary for their implementation. The most common deployment scenarios are:

1. Forward deployment to a single server
2. Forward deployment to multiple servers
3. Forward deployment of different directories to different servers
4. Reverse deployment
5. Reverting the website
6. Deploying through firewalls

Any of these types of deployment can be triggered in various ways. For example, deployment can be triggered:

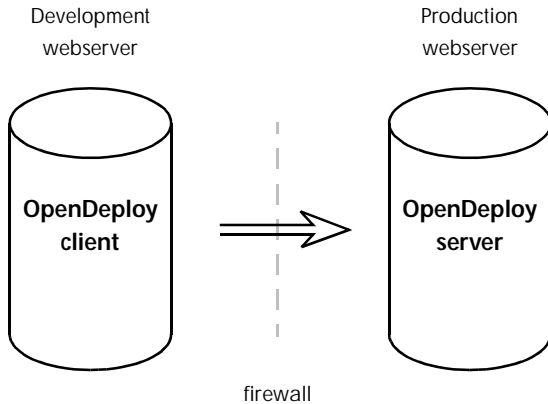
- manually
- on publication of an edition (using the TeamXpress `iwatpub` command trigger)
- on submission of a workarea (using the TeamXpress `iwatsub` command trigger)
- at a certain time (using `cron` or `at`)

In the first case, the `iwdeploy` client is called from the command line. In the other three, it is called by custom scripts. This chapter describes how to deploy manually from the command line.

The following sections include simple deployment configuration files for each of the six scenarios listed above. You can modify these configuration files to suit your individual site's needs.

Forward Deployment to a Single Server

Forward deployment takes all the files to be deployed (as specified in the client configuration file) and deploys them to the production server. The method used to determine which files to deploy is specified in the `iwdeploy` client configuration file (see page 265).



Forward deployment

This deployment requires a configuration file for the `iwdeploy` client on the development server and a configuration file for the `iwdeploy` server on the production server. For this example, the `iwdeploy` server configuration file will be named `remote_receive.cfg` and the `iwdeploy` client configuration file will be named `local_send.cfg`. The configuration files used for this example are included in the following sections.

1. To initiate deployment on a UNIX production server, issue the following command from the production server command line prompt:

```
% iwdeploy -s -fd path/remote_receive.cfg
```

Or, on a Windows NT/2000 server:

```
>iwdeploy -s -fd path\remote_receive.cfg
```

Alternatively, you can invoke the OpenDeploy server from the **Services** control panel, and type `-s -fd path\remote_receive.cfg` in the **Startup Parameters** box (see page 239).

The `iwdeploy` server will listen for incoming connections on the port specified in its configuration file (`remote_receive.cfg`).

2. On a UNIX development server, issue the following command from the command line prompt:

```
% iwdeploy -fs path/local_send.cfg deploy_to_single
```

Or, on a Windows NT/2000 development server:

```
>iwdeploy -fs path\local_send.cfg deploy_to_single
```

The `iwdeploy` client will attempt to connect to the `iwdeploy` server process on the production server. It will use the `remote_server` and `remote_port` listed in `local_send.cfg` to determine which server to connect to.

Note: the port number listed in both configuration files must be the same for the connection to be established. You may also need to open the specified port if a firewall is in place.

3. The `iwdeploy` client will read the global parameters of the configuration file, find the `deploy_to_single` deployment section of the configuration file, read its parameters, and deploy the specified content.

OpenDeploy Server Configuration

The server configuration file `remote_receive.cfg` specifies the port number that the `iwdeploy` server will listen to, and the key file for establishing a handshake. This file must be located on the production server. The following server configuration files allow you to execute forward deployment to a single server:

UNIX

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=/u/iw/andre/deploy/encryptkey
    allowed_directory = /tmp/deploydst
;
```

Windows NT/2000

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
```

```
key_file=d:\deploy\encryptkey
allowed_directory = d:\deploydst\content
;
```

OpenDeploy Client Configuration

The client configuration file `remote_send.cfg` specifies all the deployment options for the files or directories being deployed. This file must be located on the TeamXpress (development) server. The following client configuration files allow you to execute forward deployment to a single server:

UNIX

```
#-----
#
# Forward deployment to a single server
#
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709
deployment=deploy_to_single
    area=/u/iw/andre/deploy
    key_file=/u/iw/andre/deploy/encryptkey
    local_directory=deploysrc
    remote_directory=/tmp/deploydst
;
;
```

Windows NT/2000

```
#-----
#
# Forward deployment to a single server
#
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
remote_server=production1.example.com
```

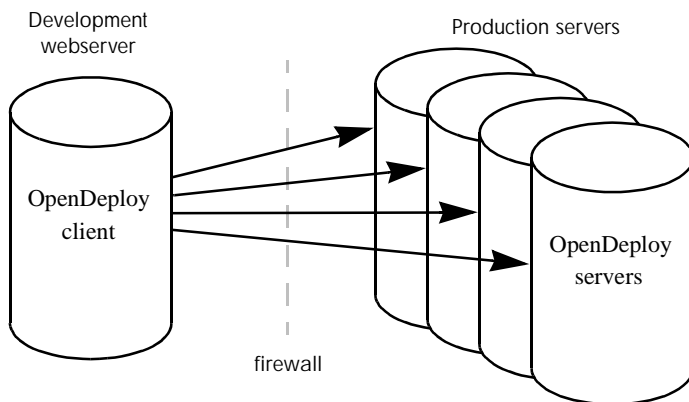
```

remote_port=1709
deployment=deploy_to_single
  area=y:\default\main\dev\EDITION
  key_file=d:\deploy\encryptkey
  local_directory=deploysrc
  remote_directory=d:\deploydst\content
;
;

```

Forward Deployment to Multiple Servers

In this deployment scenario, the `iwdeploy` client takes all the files to be deployed (as specified in the client configuration file) and deploys them to multiple production servers.



Forward deployment to multiple servers

This deployment requires a configuration file for the `iwdeploy` client on the development server and a configuration file for the `iwdeploy` server on each production server. For this example, the production host configuration files will be all be named `remote_receive.cfg` and the TeamXpress host configuration file will be named `local_send.cfg`. The configuration files used for this example are included in the following section.

1. To initiate deployment on UNIX production servers, issue the following command from the command line prompt on *each* production server:

```
% iwdeploy -S -fd path/remote_receive.cfg
```



Or, on Windows NT/2000 production servers:

```
>iwdeploy -s -fd path\remote_receive.cfg
```

Alternatively, you can invoke the OpenDeploy server from the **Services** control panel, and type `-s -fd path\remote_receive.cfg` in the **Startup Parameters** box (see page 239).

The `iwdeploy` server will listen for incoming connections on the port specified in its configuration file (`remote_receive.cfg`).

2. On a UNIX development server, issue the following commands from the command line prompt:

```
% iwdeploy -fs path/local_send.cfg deploy_to_srv1
```

```
% iwdeploy -fs path/local_send.cfg deploy_to_srv2
```

```
% iwdeploy -fs path/local_send.cfg deploy_to_srv3
```

Or, from a Windows NT/2000 development server:

```
>iwdeploy -fs path\local_send.cfg deploy_to_srv1
```

```
>iwdeploy -fs path\local_send.cfg deploy_to_srv2
```

```
>iwdeploy -fs path\local_send.cfg deploy_to_srv3
```

Alternatively, you can write a script to invoke the `iwdeploy` client.

The `iwdeploy` client will attempt to connect to the `iwdeploy` server process on the production server. It will use the `remote_server` and `remote_port` listed in `local_send.cfg` to determine which computer to connect to.

Note: the port number listed in both configuration files must be the same for the connection to be established. You might also need to open the specified port if a firewall is in place.

3. The `iwdeploy` client will read the global parameters of the configuration file, find the specified deployment sections of the configuration file, read their parameters, and deploy the specified content.

OpenDeploy Server Configuration

The server configuration file `remote_receive.cfg` specifies the port number that the `iwdeploy` server will listen to, and the key file for establishing a handshake. This file must be located on the production host. The following server configuration files allow you to execute forward deployment to multiple servers:

UNIX

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=/u/iw/andre/deploy/encryptkey
    allowed_directory = /tmp/deploydst
;
```

Windows NT/2000

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=d:\deploy\encryptkey
    allowed_directory = d:\deploydst\content
;
```

OpenDeploy Client Configuration

The client configuration file `remote_send.cfg` specifies all the deployment options for the files or directories being deployed. This file must be located on the development server. The following client configuration files allow you to execute forward deployment to multiple servers:

UNIX

```
#-----
#
# Forward deployment to multiple servers
#
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
deployment=deploy_to_srv1
    remote_server=production1.example.com
    remote_port=1709
    area=/u/iw/andre/deploy
    key_file=/u/iw/andre/deploy/encryptkey
    local_directory=deploysrc
    remote_directory=/tmp/deploydst
;
```



```
;
deployment=deploy_to_srv2
    remote_server=production2.example.com
    remote_port=1710
    area=/u/iw/andre/deploy
    key_file=/u/iw/andre/deploy/encryptkey
    local_directory=deploysrc
        remote_directory=/tmp/deploydst
    ;
;
deployment=deploy_to_srv3
    remote_server=production3.example.com
    remote_port=1711
    area=/u/iw/andre/deploy
    key_file=/u/iw/andre/deploy/encryptkey
    local_directory=deploysrc
        remote_directory=/tmp/deploydst
    ;
;
```

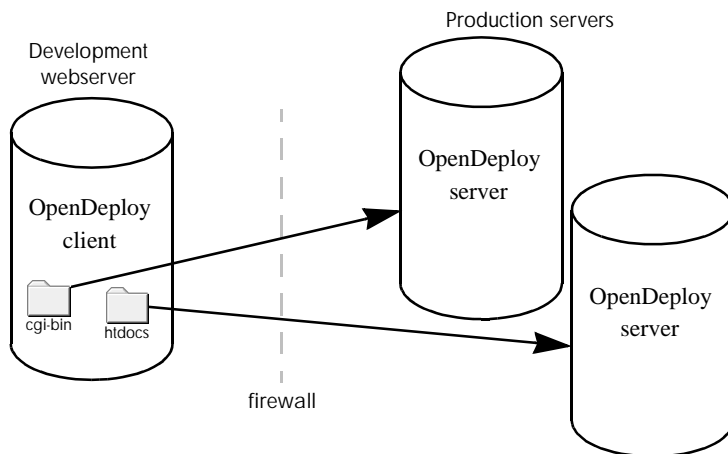
Windows NT/2000

```
#-----
#
# Forward deployment to multiple servers
#
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
deployment=deploy_to_srv1
    remote_server=production1.example.com
    remote_port=1709
    area=y:\default\main\dev\EDITION
    key_file=d:\deploy\encryptkey
    local_directory=deploysrc
        remote_directory=d:\deploydst\content
    ;
;
deployment=deploy_to_srv2
    remote_server=production2.example.com
```

```
remote_port=1710
area=y:\default\main\dev\EDITION
key_file=d:\deploy\encryptkey
local_directory=deploysrc
    remote_directory=d:\deploydst\content
;
;
deployment=deploy_to_srv3
remote_server=production3.example.com
remote_port=1711
area=y:\default\main\dev\EDITION
key_file=d:\deploy\encryptkey
local_directory=deploysrc
    remote_directory=d:\deploydst\content
;
;
```

Forward Deployment of Different Directories to Different Servers

In this scenario, the `iwdeploy` client takes the files to be deployed in certain directories on the development server (as specified in the client configuration file) and deploys them to different production servers.



Forward deployment of different directories to different servers

This deployment requires a configuration file for the `iwdeploy` client on the development server and a configuration file for the `iwdeploy` server on each production server. For this example, the server configuration files on the production servers will be all be named `remote_receive.cfg` and the client configuration file on the development server will be named `local_send.cfg`. The configuration files used for this example are included in the following section.

1. To initiate deployment on UNIX production servers, issue the following command from the command line prompt on *each* production server:

```
% iwdeploy -s -fd path/remote_receive.cfg
```

Or, on Windows NT/2000 production servers:

```
>iwdeploy -s -fd path\remote_receive.cfg
```

Alternatively, you can invoke the OpenDeploy server from the **Services** control panel, and type `-s -fd path\remote_receive.cfg` in the **Startup Parameters** box (see page 239).

The `iwdeploy` server will listen for incoming connections on the port specified in its configuration file (`remote_receive.cfg`).

2. On a UNIX development server, issue the following commands from the command line prompt:

```
% iwdeploy -fs path/local_send.cfg deploy_to_srv1
```

```
% iwdeploy -fs path/local_send.cfg deploy_to_srv2
```

Or, from a Windows NT/2000 development server:

```
>iwdeploy -fs path\local_send.cfg deploy_to_srv1
```

```
>iwdeploy -fs path\local_send.cfg deploy_to_srv2
```

Alternatively, you can write a script to invoke the `iwdeploy` client.

The `iwdeploy` client will attempt to connect to the `iwdeploy` server process on the production server. It will use the `remote_server` and `remote_port` listed in `local_send.cfg` to determine which computer to connect to.

Note: the port number listed in both configuration files must be the same for the connection to be established. You might also need to open the specified port if a firewall is in place.

3. The `iwdeploy` client will read the global parameters of the configuration file, find the appropriate deployment sections of the configuration file, read their parameters, and deploy the specified content.

OpenDeploy Server Configuration

The server configuration file `remote_receive.cfg` specifies the port number that the `iwdeploy` server process will listen to, and the key file for establishing a handshake. This configuration file must be located on the production server. The following server configuration files allow you to deploy different directories to different production servers.

UNIX

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=/u/iw/andre/deploy/encryptkey
    allowed_directory = /tmp/deploydst
;
```

Windows NT/2000

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=d:\deploy\encryptkey
    allowed_directory = d:\deploydst\content
;
```

OpenDeploy Client Configuration

The client configuration file `remote_send.cfg` specifies all the deployment options for the files or directories being deployed. This file must be located on the development server. The following client configuration files allow you to deploy different directories to different production servers.

UNIX

```
#-----
#
# Forward deployment of different directories
# to different servers
#
# iwdeploy CLIENT config file
#
```



```
#-----  
hostname=development1.example.com  
  
deployment=deploy_to_srv1  
    remote_server=production1.example.com  
    remote_port=1709  
    area=/u/iw/andre/deploy  
    key_file=/u/iw/andre/deploy/encryptkey  
    local_directory=htdocs  
        remote_directory=/tmp/deploydst1  
    ;  
;  
deployment=deploy_to_srv2  
    remote_server=production2.example.com  
    remote_port=1710  
    area=/u/iw/andre/deploy  
    key_file=/u/iw/andre/deploy/encryptkey  
    local_directory=cgi-bin  
        remote_directory=/tmp/deploydst2  
    ;  
;
```

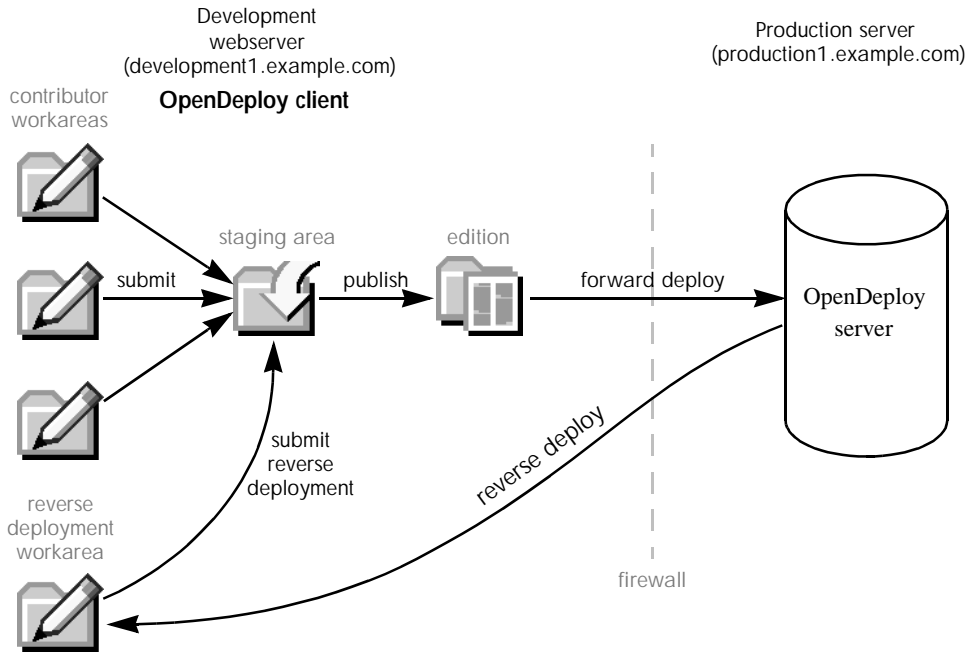
Windows NT/2000

```
#-----  
#  
# Forward deployment of different directories  
# to different servers  
#  
# iwdeploy CLIENT config file  
#  
#-----  
hostname=development1.example.com  
  
deployment=deploy_to_srv1  
    remote_server=production1.example.com  
    remote_port=1709  
    area=y:\default\main\dev\EDITION  
    key_file=d:\deploy\encryptkey  
    local_directory=htdocs  
        remote_directory=d:\deploydst\content1
```

```
    ;  
;  
deployment=deploy_to_srv2  
    remote_server=production2.example.com  
    remote_port=1710  
    area=y:\default\main\dev\EDITION  
    key_file=d:\deploy\encryptkey  
    local_directory=cgi-bin  
        remote_directory=d:\deploydst\content2  
    ;  
;
```

Reverse Deployment

Reverse deployment is used when changes are made directly to the production server and need to be brought back into TeamXpress for development and versioning. OpenDeploy checks to see which files have changed on the production server and copies them into a workarea on the development server. The files can then be submitted to the staging area so that users can check their work in the context of the staging area, and bring the changed files from the staging area into their individual workareas.



Forward deployment combined with periodic reverse deployment

In the scenario above, forward deployment is combined with a periodic reverse deployment to a workarea created for that purpose. Triggering of reverse deployment and the submission of the reverse deployed content is handled by custom scripts.

Reverse deployment requires a total of four configuration files—two on the development server and two on the production server.

The two configuration files for the `iwdeploy` client on the development server (in this example, `development1.example.com`) are:

- `local_dummy_send.cfg`
- `local_receive.cfg`

The two configuration files for the `iwdeploy` server on each production server (in this example, `production1.example.com`) are:

- `remote_receive.cfg`
- `remote_send.cfg`

The configuration files used for this example are included in the following section.

1. To initiate deployment from a UNIX production server, issue the following command from the production server command line prompt:

```
% iwdeploy -S -fd path/remote_receive.cfg -fs path/remote_send.cfg
```

Or, on a Windows NT/2000 production server:

```
>iwdeploy -S -fd path\remote_receive.cfg -fs path\remote_send.cfg
```

Alternatively, you can invoke the OpenDeploy server from the **Services** control panel, and type `-s -fd path\remote_receive.cfg -fs path\remote_send.cfg` in the **Startup Parameters** box (see page 239).

The `iwdeploy` server will listen for incoming connections on the port specified in `remote_receive.cfg`.

2. On a UNIX development server, issue the following command from the command line prompt:

```
% iwdeploy -r -fs path/local_dummy_send.cfg -fd path/local_receive.cfg  
reverse_deploy
```

Or, from a Windows NT/2000 development server:

```
>iwdeploy -r -fs path\local_dummy_send.cfg -fd path\local_receive.cfg  
reverse_deploy
```

In the preceding example, `-r` specifies a reverse deployment and `reverse_deploy` is the name of the deployment in `local_dummy_send.cfg`.

The `iwdeploy` client on the development server will attempt to connect to the `iwdeploy` server process on the production server. It will use the `remote_server` and `remote_port` definitions listed in `local_dummy_send.cfg` to determine which computer to connect to. The `iwdeploy` server will be listening to the server and port specified in `remote_receive.cfg`.

Note: the port number listed in both configuration files must be the same for the connection to be established. You might also need to open the specified port if a firewall is in place.

3. The `iwdeploy` server on the production server will recognize that a reverse deployment is underway. It will check `remote_send.cfg` to determine what files to push to the `iwdeploy` client. The `iwdeploy` client will determine where to put the files using `local_receive.cfg`.

OpenDeploy Server Configuration

`remote_send.cfg`

The server configuration file `remote_send.cfg` provides sending parameters such as source area, local and remote directories, file exclusion filters, and permission filters.

`hostname` specifies the production server that will be receiving content.

`remote_server` specifies the development server that will be sending content.

`area` specifies the directory containing the local directory of files to be sent back to the development server.

`local_directory` specifies the directory (relative to `area`) on the production server that contains the files to be sent. `local_directory` and `area` make up the full path of the directory to be sent.

`remote_directory` specifies the directory on the development server that will receive content.

The following server configuration files allow you to execute reverse deployment from the production server.

UNIX

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy SERVER source config file  
#  
#-----  
hostname=production1.example.com  
deployment=reverse_deploy  
    remote_server=development1.example.com  
    remote_port=1709  
    area=/u/iw/andre/deploy  
    key_file=/u/iw/andre/deploy/encryptkey  
    local_directory=deploysrc  
        remote_directory=/tmp/deploydst  
    ;  
;
```

Windows NT/2000

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy SERVER source config file  
#  
#-----  
hostname=production1.example.com  
deployment=reverse_deploy  
    remote_server=development1.example.com  
    remote_port=1709  
    area=y:\default\main\dev\EDITION  
    key_file=d:\deploy\encryptkey  
    local_directory=deploysrc  
        remote_directory=d:\deploydst\content  
    ;  
;
```

remote_receive.cfg

The server configuration file `remote_receive.cfg` is used to specify the port number that the `iwdeploy` server process will listen to, and the key file for establishing a handshake. `TeamSite_server` is the development server that will receive content from the production server. The following server configuration files allow you to execute reverse deployment from the production server.

UNIX

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy SERVER config file  
#  
#-----  
port=1709  
TeamSite_server=production1.example.com  
key_file=/u/iw/andre/deploy/encryptkey
```

Windows NT/2000

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy SERVER config file  
#  
#-----  
port=1709  
TeamSite_server=production1.example.com  
key_file=key_file=d:\deploy\encryptkey
```


OpenDeploy Client Configuration

`local_dummy_send.cfg`

On the development server, `local_dummy_send.cfg` specifies the port number that the `iwdeploy` client will attempt to contact on the production server. `hostname` specifies the development server, and `remote_server` specifies the production server.

Also, a dummy deployment must be specified. This dummy deployment name will be passed to the production host and must match a valid deployment name in the production host's `remote_send.cfg`. A dummy area for the dummy deployment is also needed, although it is disregarded.

`hostname` is the name of the development server.

`remote_server` is the name of the production server.

`remote_port` must be the same port number that is specified in all the other configuration files.

`deployment` is the name of the dummy deployment.

`area` is the name of a directory. Although this directory is not used in deployment, it must be a valid directory.

`key_file` is optional, but if it is included in one of the four configuration files, it must be included in them all, and it must match the key file specified in the `iwdeploy` server configuration files. The following client configuration files allow you to execute reverse deployment from the production server.

UNIX

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy CLIENT 'dummy' config file  
#  
#-----  
hostname=development1.example.com  
remote_server=production1.example.com  
remote_port=1709  
  
deployment=reverse_deploy  
    area=/u/iw/andre/deploy  
    key_file=/u/iw/andre/deploy/encryptkey  
;
```

Windows NT/2000

```
#-----  
#  
# Reverse deployment  
#  
# iwdeploy CLIENT 'dummy' config file  
#  
#-----  
hostname=development1.example.com  
remote_server=production1.example.com  
remote_port=1709  
  
deployment=reverse_deploy  
    area=y:\default\main\dev\EDITION  
    key_file=d:\deploy\encryptkey  
;
```

local_receive.cfg

Because the development server will receive files, `local_receive` is needed only to specify:

- which servers are allowed to deploy to the development server,
- the key file, and
- a list of allowed directories that can be deployed to on the local host.

`TeamSite_server` specifies the production server that will send content to the development server.

`key_file` (if included) must match the key file specified in all the other configuration files.

`allowed_directory` specifies the directory on the development server that will receive the content.

The following client configuration files allow you to execute reverse deployment from the production server.

UNIX

```
#-----
# Reverse deployment
# iwdeploy CLIENT destination config file
#-----
port=1709
TeamSite_server=production1.example.com
    key_file=/u/iw/andre/deploy/encryptkey
    allowed_directory = /tmp/deploydst
;
```

Windows NT/2000

```
#-----
# Reverse deployment
# iwdeploy CLIENT destination config file
#-----
port=1709
TeamSite_server=production1.example.com
    key_file=d:\deploy\encryptkey
    allowed_directory = d:\deploydst\content
;
```

Reverting Websites to Previous Versions

OpenDeploy uses Interwoven Site Rollback technology to allow users to revert the production server to an earlier version of the website. OpenDeploy uses directory comparison (see page 265) to compare the files in the deployment directory on the development server and the directory being deployed to on the production server. It then deploys the files that have older timestamps.

This deployment requires a configuration file for the `iwdeploy` client on the development server and a configuration file for the `iwdeploy` server on the production server. For this example, the server configuration file will be named `remote_receive.cfg` and the client configuration file on the development server will be named `local_send.cfg`. The configuration files used for this example are included in the following sections.

1. To initiate deployment on a UNIX production server, issue the following command from the production server command line prompt:

```
% iwdeploy -s -fd path/remote_receive.cfg
```

Or, on a Windows NT/2000 production server:

```
>iwdeploy -s -fd path\remote_receive.cfg
```

Alternatively, you can invoke the OpenDeploy server from the **Services** control panel, and type `-s -fd path\remote_receive.cfg` in the **Startup Parameters** box (see page 239).

The `iwdeploy` server will listen for incoming connections on the port specified in its configuration file (`remote_receive.cfg`).

2. On a UNIX development server, issue the following command from the command line prompt:

```
% iwdeploy -fs path/local_send.cfg revert
```

Or, from a Windows NT/2000 development server:

```
>iwdeploy -fs path\local_send.cfg revert
```

The `iwdeploy` client will attempt to connect to the `iwdeploy` server process on the production server. It will use the `remote_server` and `remote_port` listed in `local_send.cfg` to determine which computer to connect to.

Note: the port number listed in both configuration files must be the same for the connection to be established. You might also need to open the specified port if a firewall is in place.

3. The OpenDeploy client will read the global parameters of the configuration file, find the `revert` deployment section of the configuration file, read its parameters, and deploy the specified content.

OpenDeploy Server Configuration

The server configuration file `remote_receive.cfg` specifies the port number that the `iwdeploy` server process will listen to, and the key file for establishing a handshake. This configuration file must be located on the production server. The following server configuration files allow you to execute Site Rollback.

UNIX

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=/u/iw/andre/deploy/encryptkey
    allowed_directory = /tmp/deploydst
;
```

Windows NT/2000

```
# Basic iwdeploy SERVER configuration file
port=1709
TeamSite_server=development1.example.com
    key_file=d:\deploy\encryptkey
    allowed_directory = d:\deploydst\content
;
```

OpenDeploy Client Configuration

The client configuration file `remote_send.cfg` specifies all the deployment options for the files or directories being deployed. This configuration file must be located on the development server. The following client configuration files allow you to excute Site Rollback.

UNIX

```
#-----
#
# Revert the website
#
```



```
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709

deployment=revert
  area=/u/iw/andre/deploytest
  key_file=/u/iw/andre/deploytest/encryptkey
  revert
  local_directory=deploysrc
    remote_directory=/tmp/deploydst
  ;
;
```

Windows NT/2000

```
#-----
#
# Revert the website
#
# iwdeploy CLIENT config file
#
#-----
hostname=development1.example.com
remote_server=production1.example.com
remote_port=1709

deployment=revert
  area=y:\default\main\dev\EDITION
  key_file=d:\deploy\encryptkey
  revert
  local_directory=deploysrc
    remote_directory=d:\deploydst\content
  ;
;
```

Deploying Through Firewalls

To deploy your website through a firewall:

1. Open the outbound port specified in the `iwdeploy` client and server configuration files.
2. Invoke your deployment.
3. (Optional) Close the port.

If you cannot open a port:

1. Use the `iwdeploy` client's `deployment_package` option (`-o`) to create a deployment package:

```
% iwdeploy -o packagename -fs srcConfigFile deployment_name
```
2. Transfer the package to the production server by the method of your choice.
3. Use the `iwdeploy` server's `deployment_package` option (`-i`) to deploy the deployment package:

```
% iwdeploy -S -i packagename -fd destConfigFile
```


Section 4: Appendices

-
- Creating Data Capture Templates from DTDs
 - Using Command-Line Tools
 - DataDeploy Database Auto-Synchronization
 - DataDeploy Database Server Configuration
 - DataDeploy Querying Tables
 - OpenDeploy Client and Server Configuration File Options

Creating Data Capture Templates from DTDs

You can create `datacapture.cfg` files that define data capture templates (DCTs) from industry-standard XML DTDs. These data capture templates display as data capture forms in TeamXpress Templating. A list of the steps to convert DTDs is outlined here. Refer to the remainder of this appendix for details and examples of the files at each step in the processing of creating the `datacapture.cfg` file.

1. Verify that the DTD is correct.
2. Run the `iwtdtd2sym` CLT to convert the DTD.
3. Copy the output from the `iwtdtd2sym` CLT to `symbol-table.cfg`.
4. Optionally modify the `symbol-table.cfg` to change the `name` attribute of the `itemref` subelement of the `<ruleset>`, and save the file.
5. Make any additional edits to add items such as labels and descriptions to `<items>` and save the file.
6. Run the `iwsym2dct` CLT.
7. Copy the output file to `datacapture.cfg`.
8. Identify the new `datacapture.cfg` file in the `templating.cfg` file.

Be sure to save all your intermediate output files along with the DTD and the final `datacapture.cfg` file. It is recommended that these files be versioned in TeamXpress.

The file will contain `<symbol>` elements that define the elements from the DTD. Refer to the Symbol Table DTD in “Symbol Table DTD Used for Conversions” on page 361 for information on the `<symbol-table>` and `<symbol>` elements.

In this final file, `<symbol>` elements have been changed to `<item>` elements.

Running the CLT on the DTD File

The following file is a sample DTD, named `simple.dtd`.

```
<!-- This is a simple example DTD.  
      It is a "Hello, world!" type of DTD.  
-->  
  
<!ELEMENT simple-example (message)>  
      <!ATTLIST simple-example  
        color      (red|blue|green)      #IMPLIED  
      >  
  
<!ELEMENT message      (#PCDATA)>
```

Run the `iwtdtd2sym` CLT on the DTD, specifying the complete path to the DTD, to create the file that begins on the next page by changing to the directory containing the DTD:

```
cd Y:\default\main\WORKAREA\chris\templatedata\internet\simple-example
```

(the reference to the Y: drive is not needed for Solaris platforms) and issuing the command:

```
iwtdtd2sym simple.dtd > iwtdtd2sym.out
```

Refer to the Appendix B, “Using Command-Line Tools” for additional details on `iwtdtd2sym`.

The `symbol-table.cfg` File

The following file is the output from the `iwtdtd2sym` CLT (`iwtdtd2sym.out`), which has been copied to a file named `symbol-table.cfg` and then edited.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data-capture-requirements SYSTEM "datacapture4.5.dtd">

<data-capture-requirements dtd-system-identifier="simple.dtd" name=""
type="content">
  <symbol-table>
    <symbol name="simple-example">

      <container combination="and" hide-name="f" name="simple-example">

        <itemref name="{iw_attributes}"/>
        <container combination="and" hide-name="t"
name="{iw_sub_elements}[0]">
          <label>XML sub-elements</label>
          <itemref name="message"/>
        </container>
      </container>
    </symbol>

    <symbol regex="^(.*)?simple-example/{iw_attributes}$">

      <container combination="and" hide-name="t"
name="{iw_attributes}">
        <item name="color">
          <select multiple="f" required="f" size="0" width="0">
            <option label="red" selected="f" value="red"/>
            <option label="blue" selected="f" value="blue"/>
            <option label="green" selected="f" value="green"/>
          </select>
        </item>
      </container>
    </symbol>
```

Reference to simple.dtd maintained. ¹

XML elements entered as symbols. ²

A <container> contains an <itemref>. ³

A set of attributes also become a <symbol>. ⁴

The color attribute. ⁵



The message element as a
#PCDATA item. ⁶

```
<symbol name="message">  
  <container combination="and" hide-name="f" name="message">  
    <container combination="and" hide-name="t"  
      name="{iw_sub_elements}[0]">  
      <label>XML sub-elements</label>  
      <item name="#PCDATA">  
        <textarea cols="0" required="f" rows="0" rtf="f"  
          wrap="off"/>  
      </item>  
    </container>  
  </container>  
</symbol>  
</symbol-table>
```

Editing the ruleset name. ⁷

```
<ruleset name="This is my only rule">  
  <itemref name="simple-example"/>  
</ruleset>
```

```
</data-capture-requirements>
```

Diagram Key

1. This file maintained a reference to the DTD from which it originated, in the `dtc-system-identifier` attribute of the `data-capture-requirements` element.
2. This file was generated directly from an industry-standard XML DTD. Each XML element becomes a `<symbol>` element in the `<symbol-table>`.
3. Every element type declared in the DTD is represented in its `<symbol>` as a `<container>`. A `<container>` that represents an XML element type will contain an `<itemref>` element for the element type's attributes, if any. A `<container>` that represents an XML element type will contain another `<container>` for its subelements. A `<container>` that represents a set of the subelements of an XML element type will contain an `<itemref>` reference for each subelement type it refers to. This XML element type (`simple-example`) has a simple content specification (`message`), so there is just one `<itemref>`.
4. The set of attributes for each element type also becomes a `<symbol>`.
5. The set of attributes of an element type is represented in its `<symbol>` as a `<container>`. There is only one attribute, `color`. Because it was an enumerated attribute, it is represented here by a `<select>` element.
6. Here is the `message` element type. Its content specification was also simple: `#PCDATA`. A character data reference in the DTD is transformed into a data capture `<item>` named `#PCDATA`.
7. The `name` attribute of the `itemref` subelement of the `<ruleset>` defaults to the name of the first element type declared in the DTD. The `ruleset` contains a single `<itemref>`. The `itemref` name defaults to the `symbol` name. This `<itemref>` references the outermost element of the XML documents that will be generated as DCRs. In this example, the `ruleset` name was edited.

You may manually add items such as labels and descriptions to this file. Examples of edits you may want to make would be to add `<label>` and `<description>` elements to `<items>` and `<containers>` and to specify `<min>` and `<max>` values in a `<replicant>` element.

The datacapture.cfg File

The next step is to run the `iwsym2dct` CLT on the edited `symbol-table.cfg` file.

Run the `iwsym2dct` CLT by issuing the command:

```
iwsym2dct symbol-table.cfg > iwsym2dct.out
```

Refer to Appendix B, “Using Command-Line Tools” for additional details on `iwsym2dct`.

Copy `iwsym2dct.out` to `datacapture.cfg`. A sample `datacapture.cfg` file follows.


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE data-capture-requirements SYSTEM "datacapture4.5.dtd">

<data-capture-requirements dtd-system-identifier="simple.dtd"
name="" type="content">
  <ruleset name="This is my only rule">
    <label>This is my only rule</label>
    <container combination="and" hide-name="f" name="simple-example">
      <label>simple-example</label>
      <container combination="and" hide-name="t"
name="{iw_attributes}">
        <label>{iw_attributes}</label>
        <item name="color">
          <label>color</label>
          <select multiple="f" required="f" size="0" width="0">
            <option label="red" selected="f" value="red"/>
            <option label="blue" selected="f" value="blue"/>
            <option label="green" selected="f" value="green"/>
          </select>
        </item>
      </container>
      <container combination="and" hide-name="t"
name="{iw_sub_elements}[0]">
        <label>XML sub-elements</label>
        <container combination="and" hide-name="f" name="message">
          <label>message</label>
          <container combination="and" hide-name="t"
name="{iw_sub_elements}[0]">
            <label>XML sub-elements</label>
            <item name="#PCDATA">
              <label>#PCDATA</label>
              <textarea cols="0" required="f" rows="0" rtf="f"
wrap="off"/>
            </item>
          </container>
        </container>
      </container>
    </container>
  </ruleset>

```

Reference to simple.dtd maintained.

The added <ruleset>. ²

{iw_attributes} in <container>. ³

(message) in <container>. ⁴

Diagram Key

1. Notice that the name of the original DTD has made it all the way to this `datacapture.cfg` file. That `dtd-system-identifier` will be used in the document type declaration of data content records generated from this DCT.
2. The `<ruleset>` used to contain a single `<itemref>`, referring to `simple-example`. That reference was expanded, using the symbol table, into a `<container>`.
3. The reference to `{iw_attributes}` was expanded into a `<container>`.
4. The reference to `message` expanded into a `<container>`.

Unsupported DTD Features

A few features in DTDs are not supported by the CLTs and the conversion process:

- An element `<section>` that can legally, according to the DTD, contain another `<section>` element is not supported to an arbitrary depth. The data capture template author must decide the depth to which an element can recursively contain elements of the same type. This is done with a regex on the `<symbol>` element.

An example of this section of a DTD is:

```
<!ELEMENT body (section)*>
<!ELEMENT section (title|paragraph|sub-section)*>
<!ELEMENT subsection (section)*>
...
```

The following is an example of a regex that captures a `<section>` element with another `<section>` element:

```
<symbol regex="^(.*)?section/(.*)?section$">
```

The following is an example of a regex that captures a `<section>` element with another `<section>` element within another `<section>` element:

```
<symbol regex="^(.*)?section/(.*)?section/(.*)?section$">
```

The first regex would catch a two-deep nesting level (anything greater than or equal to two), and the second regex would catch any nesting level 3 or greater. The regex `<symbol name="section">` would catch all levels.

Therefore, these symbols need to be ordered in the `symbol-table` by depth, e.g.:

```
<symbol regex="^(.*)?section/(.*)?section/(.*)?section$" >
  ...
</symbol>
<symbol regex="^(.*)?section/(.*)?section$" >
  ...
</symbol>
<symbol name="section" >
  ...
</symbol>
```

- The validity constraints for the `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN`, and `NMTOKENS` attribute types are not enforced.

For an explanation of the validity constraints, refer to section 3.3.1 of the XML 1.0 specification, located at <http://www.w3.org/TR/1998/REC-xml-19980210>.

Symbol Table DTD Used for Conversions

The `symboltable4.5.dtd` file is the DTD file that reflects the element types used when industry standard DTDs are converted to `datacapture.cfg` files. The parameter entities in the `symboltable4.5.dtd` are different from the parameter entities in the `datacapture4.5.dtd`. Additionally, the element types `<symbol-table>`, `<symbol>`, and `<itemref>` apply only to the `symboltable4.5.dtd`.

```
<!-- symboltable4.5.dtd -->
```

```
<!-- Start with some basic parameter entities. -->
```

```
<!ENTITY % data-capture-requirements-contentspec
  symbol-table,ruleset*" >
```

```
<!ENTITY % items "container|item|itemref" >
```

```
<!ENTITY % chooser-options "option" >
```

```
<!--These next three element types are specific to symboltable4.5.dtd.-->
```

```

<!ELEMENT symbol-table    (symbol*) >

<!ELEMENT symbol          (%items;)? >
  <!ATTLIST symbol
    name                CDATA                #IMPLIED
    regex               CDATA                #IMPLIED
  >

<!ELEMENT itemref        EMPTY >
  <!ATTLIST itemref
    name                CDATA                #REQUIRED
  >

<!-- The rest of these elements are common to both
      datacapture4.5.dtd and symboltable4.5.dtd. -->

<!ELEMENT data-capture-requirements
          (%data-capture-requirements-contentspec;)>
  <!ATTLIST data-capture-requirements
    name                CDATA                #IMPLIED
    type                (metadata|content|workflow) #REQUIRED
    dtd-system-identifier CDATA                #IMPLIED
  >

<!-- The 'dtd-system-identifier' attribute is a URI indicating the
      DTD from whence a particular data-capture-requirements was
      derived, if any.

      In TeamXpress Templating, the value of this attribute is used as
      the system identifier of the document type declaration of a DCR
      if and only if that DCR's type is "xml", as defined in
      templating.cfg.
      -->

<!ELEMENT ruleset        (label?,description?,(%items;)* ) >
  <!ATTLIST ruleset
    name                CDATA                #REQUIRED
  >

<!ELEMENT container      (label?,description?,(%items;)* ) >
  <!ATTLIST container

```

```

        name          CDATA          #REQUIRED
        hide-name     (t|f)         "f"
        combination   (and|or)      "and"
    >

<!ELEMENT item      (label?,description?,database?,(checkbox|radio|
                    text|textarea|select|replicant|browser|
                    readonly|hidden)+) >
    <!ATTLIST item
        name          CDATA          #REQUIRED
    >

<!ELEMENT label     (#PCDATA) >
<!ELEMENT description (#PCDATA) >

<!ELEMENT readonly  (allowed?,callout?) >

<!ELEMENT hidden    (allowed?,callout?) >
    <!ATTLIST hidden
        required      (t|f)         "f"
    >

<!ELEMENT text      (allowed?,callout?,default?) >
    <!ATTLIST text
        required      (t|f)         "f"
        maxlength     CDATA          "0"
        size          CDATA          "0"
        validation-regex CDATA      #IMPLIED
    >

<!-- validation-regex is a Perl regex for validating this element -->
<!ELEMENT textarea  (allowed?,callout?,default?) >
    <!ATTLIST textarea
        required      (t|f)         "f"
        rows          CDATA          "0"
        cols          CDATA          "0"
        wrap          (off|virtual|physical) "off"
        validation-regex CDATA      #IMPLIED
        rtf           (t|f)         "f"
    >

<!-- validation-regex is a Perl regex for validating this element -->

```



```
<!ELEMENT browser      (allowed?,callout?) >
  <!ATTLIST browser
    required      (t|f)          "f"
    maxlength     CDATA          "0"
    size          CDATA          "0"
    initial-dir   CDATA          #IMPLIED
    ceiling-dir   CDATA          #IMPLIED
    extns         CDATA          #IMPLIED
  >

<!ELEMENT checkbox     (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST checkbox
    required      (t|f)          "f"
    delimiter     CDATA          ", "
  >

<!ELEMENT radio        (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST radio
    required      (t|f)          "f"
  >

<!ELEMENT select       (allowed?,callout?,(%chooser-options;)+) >
  <!ATTLIST select
    required      (t|f)          "f"
    size          CDATA          "0"
    multiple      (t|f)          "f"
    delimiter     CDATA          ", "
    width         CDATA          #IMPLIED
  >
<!-- The delimiter attribute is for multiple=t only -->

<!ELEMENT replicant    (allowed?,(%items;)* ) >
  <!ATTLIST replicant
    min           CDATA          "0"
    max           CDATA          "1"
    default       CDATA          "1"
    combination   (and|or)       "and"
    hide-name     (t|f)          "f"
  >

<!ELEMENT option       EMPTY >
```

```

        <!ATTLIST option
            selected      (t|f)           "f"
            value         CDATA           #IMPLIED
            label         CDATA           #REQUIRED
        >

<!ELEMENT allowed      (cred|and|or|not) >

<!ELEMENT cred        EMPTY >
    <!ATTLIST cred
        role            CDATA           #IMPLIED
        user            CDATA           #IMPLIED
    >

<!ELEMENT and         ((cred|and|or|not)+) >

<!ELEMENT or          ((cred|and|or|not)+) >

<!ELEMENT not         (cred|and|or|not) >

<!ELEMENT default     (#PCDATA) >

<!ELEMENT callout     (param*) >
    <!ATTLIST callout
        type            (java-class)    #REQUIRED
        label           CDATA           #REQUIRED
        location        CDATA           #REQUIRED
        class           CDATA           #REQUIRED
    >

<!ELEMENT param       EMPTY >
    <!ATTLIST param
        name            CDATA           #REQUIRED
        value           CDATA           #REQUIRED
    >

```



```
<!ELEMENT databaseEMPTY >  
  <!ATTLIST database  
    deploy-column      (t|f)          "t"  
    searchable        (t|f)          "t"  
    data-type         CDATA          "VARCHAR ( 255 ) "  
    data-format       CDATA          #IMPLIED  
  >
```


Using Command-Line Tools

You can generate or regenerate HTML files from the command line as well as from the TeamXpress Templating GUI. Refer to the *TeamXpress User's Guide* for information on the GUI.

Both `iwgen` and `iwregen` make use of an underlying low-level presentation template compiler, called `iwpt_compile.ipl`. This compiler is available for your use and is especially beneficial when you are developing, testing, and debugging presentation templates.

The presentation template compiler, `iwpt_compile.ipl`, is a command-line tool that uses the data content records, Perl code, and `iw_xml` tags to produce output. You can use the presentation template compiler when you are developing new tags.

The `iwtdtd2sym` and `iwsym2dct` CLTs are used to create data capture templates from industry-standard DTDs. Refer to Appendix A, “Creating Data Capture Templates from DTDs” for examples of using these CLTs.

The `iwxml_validate.ipl` CLT validates XML files against a DTD.

The `upgrade_dct_cfg.ipl` CLT upgrades `datacapture.cfg` files from regex5 basic regular expression syntax to extended regular expressions.

iwdtaclevel

Alters a data capture template to have only one data capture instance per item, according to ACLs in the data capture template (DCT). It evaluates ACLs (set with `<allowed>` tags) inside DCTs. It also runs server-side callouts. The templating Java client receives a DCT from the TeamXpress server. The document it receives has been through this ACL evaluation process and the server-side inline callout substitutions. This CLT is a debugging tool that lets you see the exact DCT that the client sees, which is not the exact DCT that is in the user's workarea.

Usage:

```
iwdtaclevel [-h|-v][-c] [-e] -u username -r userrole -w workarea dct
```

Options:

<code>-h</code>	Displays this usage message.
<code>-v</code>	Displays version number.
<code>-c</code>	Displays the Java class path.
<code>-e</code>	Sends errors to STDOUT.
<code>-u <i>username</i></code>	Specifies the name of the current data capture end user.
<code>-r <i>userrole</i></code>	Specifies the role of the current data capture end user.
<code>-w <i>workarea</i></code>	Specifies a vpath to the current workarea.
<code><i>dct</i></code>	Specifies a file-system path to the current data capture template.

Example:

A data capture template that contains the following section is used:

```
<item name="just chris and andre">
  <textarea><allowed><cred user="chris" /></allowed></textarea>
  <text><allowed><cred user="andre" /></allowed></text>
</item>
```

The CLT:

```
iwidctacleval -u chris -r editor /default/main/dev/WORKAREA/chris /  
path_to/datacapture.cfg
```

issues the following results for this section:

```
<item name="just chris and andre">  
  <textarea><allowed><cred user="chris" /></allowed></textarea>  
</item>
```

However, if you issue the CLT as follows:

```
iwidctacleval -u andre -r editor /default/main/dev/WORKAREA/chris /  
path_to/datacapture.cfg
```

the following results are obtained for this section:

```
<item name="just chris and andre">  
  <text><allowed><cred user="andre" /></allowed></text>  
</item>
```

iwtdtd2sym

Converts an XML DTD into a skeletal data capture symbol table configuration file. This output must be manually modified before further use. The symbol table configuration file will be written to standard output.

Usage:

```
iwtdtd2sym [-h|-v] [-c] [-r ruleset-name] [-i itemref-name] dtd-location
```

<code>-h</code>	Displays this usage information.
<code>-v</code>	Displays this command's version number.
<code>-c</code>	Displays the Java class path.
<code>-r <i>ruleset-name</i></code>	Specifies the name of the ruleset in the outputted symbol table configuration file. Default is TeamXpress Templating.
<code>-i <i>itemref-name</i></code>	Specifies the name of the itemref in the ruleset in the outputted symbol table configuration file. Default is the name of the first element type declared in the DTD.
<code><i>dtd-location</i></code>	Specifies a system literal, which is a URI referencing an XML DTD. Example URIs are: document.dtd (a file system path) ../path/to/document.dtd (a file system path) http://www.flixml.org/flixml/flixml.dtd (a URL)

Example:

The following line converts the `simple.dtd` file in and outputs it to `iwtdtd2sym.out`.

```
iwtdtd2sym simple.dtd > iwtdtd2sym.out
```

iwgen

Generates an HTML file based on a presentation template and a data content record.

Usage:

```
iwgen [-h|-v] -t templatevpath -r recordvpath vpath
```

Options:

-h	Displays this usage message.
-v	Displays version number.
-t <i>templatevpath</i>	Specifies a path to a TeamXpress Templating presentation template, where <i>templatevpath</i> is either a relative vpath or an archive-rooted vpath. Server-rooted vpaths are not supported.
-r <i>recordvpath</i>	Specifies a path to a TeamXpress Templating data content record, where <i>recordvpath</i> is either a relative vpath or an archive-rooted vpath. Server-rooted vpaths are not supported.
<i>vpath</i>	Specifies a path to write the TeamXpress Templating generated file, where <i>vpath</i> is either a relative vpath or an archive-rooted vpath. Server-rooted vpaths are not supported.

Example:

The following example generates an HTML file based on the presentation template `auction.tpl` and the data content record `june_items`. The HTML file is written to the file `june_display.html` in the current workarea. The current working directory is the user's workarea. You should enter this as a single line.

```
% iwgen -t templatedata/internet/auction/presentation/auction.tpl  
templatedata/internet/auction/data/june_items june_display.html
```

iwpt_compile.ipl

Invokes the command-line presentation template compiler.

Usage:

```
iwpt_compile.ipl -pt filename [-ofile filename] [-ocode filename]  
[-oenc encoding] [-smartwrite] [tag-specific flags]
```

```
iwpt_compile.ipl -v | -h
```

Arguments:

<code>-v</code>	Prints the version number on STDOUT.
<code>-h</code>	Prints a help message.
<code>-pt <i>filename</i></code>	Use the <i>filename</i> presentation template.
<code>-ofile <i>filename</i></code>	Save the output to <i>filename</i> instead of STDOUT.
<code>-ocode <i>filename.ipl</i></code>	Writes to a stand-alone program named <i>filename.ipl</i> that generates the output.
<code>-oenc <i>encoding</i></code>	Specifies output encoding, which is UTF-8 by default. Specify <code>-oenc</code> on the XML declaration line of the presentation template.
<code>-smartwrite</code>	Specifies <code>-ofile</code> only overwrites <i>filename</i> if it is different.

Tag-specific flags:

<code>-iw_pt-dcr</code>	The file names that follow this <code>iwpt_compile.ipl</code> flag must be a valid data content record. <code>iw_pt</code> reads in the data content record and makes its values available through <code>iw_value</code> .
<code>-iw_pt-arg</code>	The key, value pairs that follow this flag are used to initialize the presentation template arguments within the template. This is useful when debugging a component that normally gets its <code>%iw_arg</code> initialized

`-iw_include-location`

by the `%iw_param` of its enclosing template's `<iw_include>` tag.

Mandatory when the `mode` attribute of the `iw_include` tag is `docroot`. The file path is prepended to the file name provided in the `file` attribute to form a complete file path (used to virtualize the inclusion).

Example 1

This compilation line uses `iw_pt-dcr` to obtain data from a single data content record named `x.dcr`.

```
iwpt_compile.ipl -pt xxx.iwpt -iw_pt-dcr x.dcr -ofile xxx.html
```

Example 2

This example uses `iw_pt-arg` to initialize presentation template arguments.

```
iwpt_compile.ipl -pt x.pt -iw_pt-arg k1=v1 k2="val 2" ...
```

causes `$iw_arg{k1}` to be set to `v1` and `$iw_arg{k2}` to be set to `val 2`.

Therefore, in a template you could say:

```
<iw_value name="$iw_arg{k1}"/> and you would get v1.
```

Example 3

This example uses the `-iw_include-location` flag.

```
iwpt_compile -iw_include-location /x/y/z ...other flags/args...
```



The limitations to using `iwpt_compile.ipl` directly are:

- Output pages are not associated with data content records.
- The output pages are editable pages (using SmartContext Editing) but they cannot be accessed through the TeamXpress Templating GUI.

When you call the presentation template compiler, you can specify command line arguments and flags. Command-line flags are specific to and used by various `iw_xml` tags rather than being used directly by the compiler. They are specified as part of the `iwpt_compile.ipl` command.

When a presentation template is processed from the presentation template compiler, the following steps are performed:

1. The presentation template is compiled using the command-line utility `iwpt_compile.ipl`. It may use zero or one XML-based data content records.
2. An XML parser reads the presentation template. As the parser reads, it encounters XML tags.
3. A tag object of the appropriate type is created and the parser calls that object's member functions, passing it relevant information, such as attribute list key, value data.
4. The tag object's member function emits a snippet of Perl.
5. Collectively, all the snippets of Perl that these tag object member functions emit as the parser scans the template from a program.
6. This program runs, and the result is the document (typically HTML) that merges content with look-and-feel instructions.

iwregen

Regenerates an HTML file that was generated by TeamXpress Templating based on a presentation template and a data content record. Use this command to update a generated HTML file if the presentation template or data content record that the file is based on was modified.

Usage:

```
iwregen [-h|-v] vpath
```

Options:

-h	Displays this usage message.
-v	Displays version number.
vpath	Specifies the path to the file that will be regenerated, where <i>vpath</i> is either a relative vpath or an archive-rooted vpath. Server-rooted vpaths are not supported.

Example:

The following example regenerates the HTML file `june_display.html`, which resides in the current workarea.

```
% iwregen june_display.html
```

iwsym2dct

Transforms a data capture symbol table into a data capture template (DCT). The DCT will be written to standard output.

Usage:

```
iwsym2dct [-h|-v] [-c] symbol-table
```

<code>-h</code>	Displays this usage information.
<code>-v</code>	Displays this command's version number.
<code>-c</code>	Displays the Java class path.
<code><i>symbol-table</i></code>	Specifies a file containing a data capture symbol table.

Example:

The following command converts a symbol table into a data capture template:

```
iwsym2dct symbol-table.cfg > iwsym2dct.out
```

iwxml_validate.ipl

Validates a list of XML files against a DTD (and can also check to see if the XML files are well-formed).

Usage:

```
iwxml_validate.ipl [-max_errors n] [-d level] [-well] x.xml [y.xml [...]]
```

```
iwxml_validate.ipl -h |-v
```

<code>-max_errors n</code>	Displays maximum of <i>n</i> errors before quitting XML validation on the current file. The default is to report all errors.
<code>-d level</code>	Sets debug verbosity level (where <i>level</i> is 0-3); the default debug verbosity level is 2.

Debug level

Displays

0	Nothing
1	A terse message on failure
2	Parsing warnings and failures
3	Messages on success and failure

<code>-well</code>	Checks to see if XML is well-formed, but does not validate.
<code>-h</code>	Displays this usage information.
<code>-v</code>	Displays this command's version number.

Example:

Given an XML file (e.g., `x.xml`):

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



```
<b p='c'>this</b>
<b p='a'>is</b>
<b p='zzzzzz'>a valid</b>
<b p='b'>xml file</b>
</a>
```

and a DTD (e.g., x.dtd):

```
<!ELEMENT a (b*)>
<!ELEMENT b (#PCDATA)>
<!ATTLIST b p CDATA #REQUIRED>
```

the command line:

```
iwxml_validate.ipl x.xml
```

will return with no output and an exit status indicating success since x.xml is a valid XML file.

upgrade_dct_cfg.ipl

The `upgrade_dct_cfg.ipl` CLT upgrades `datacapture.cfg` files from regex5 basic regular expression syntax to extended regular expressions. The meanings of the original basic regular expressions are preserved, but the extended regex grammar provides more expressive power for validating user input.

This upgrade is required when moving from the browser-based data capture interface of TeamXpress Templating to the Java-based interface; however, you can use extended regular expressions in the browser-based interface. Only validation-regex attributes containing the following characters are affected: `+ ? | ()`

Usage:

```
upgrade_dct_cfg.ipl [-log file] [-inplace] [-n] [-d verbosity]
[-no_iwcfg_update] [-force] [-no_staging_update]
[directory_name|file_name]+
```

```
upgrade_dct_cfg.ipl -v | -h
```

`-log file`

The name of the file to which log information is sent. By default, log information is printed on STDOUT. For example, if `-log xxx` is used, all log information is sent to the file named `xxx`.

`-inplace`

Do not make backup copies of the `datacapture.cfg` files; without this switch, `datacapture.cfg.backup` files are placed in the same directories as the `datacapture.cfg` files.

`-n`

Do not write or modify any `datacapture.cfg` files; just determine which ones require an upgrade. Do not modify `/etc/iw.cfg`.

`-no_iwcfg_update`

Do not modify anything in `iw.cfg`. By default, running this utility sets `use_extended_regex5=true` within the `[teamsite_templating]` section of `/etc/iw.cfg`.



`-force`

This utility should run at most once on the root of TeamXpress branching structure (e.g., `/iwmnt`) since the conversion from basic regexes to extended regexes is one-way. If

`use_extended_regex5=true` is already set within the `[teamsite_templating]` section of `iw.cfg`, it is assumed that no further conversion of `datacapture.cfg` files is required, and this utility will exit with a diagnostic message. To override this behavior, use the `-force` flag.

`-no_staging_update`

Do not attempt to upgrade `datacapture.cfg` files that are already in the staging area. By default, `datacapture.cfg` files in the staging area are upgraded by creating a temporary workarea, doing an update of the relevant files, and then automatically checking in the changes.

`-d verbosity`

Set the debug verbosity level:

Verbosity Displays

- 0 Nothing
- 1 Only files requiring upgrade
- 2 Changed and unchanged files (default)
- 3 Information from Level 2 plus low-level trade messages
- 4 Information from Level 3 with extensive trace messages

`-h`

Displays this usage information.

`-v`

Displays this command's version number.

Example:

```
upgrade_dct_cfg.ipl $iwmount
```

CAUTION: You should not run this utility more than once on `$iwmount` (see `-force` for details).

Background:

In *basic* regular expressions (the old default):

Character Meaning

+	a single instance of the '+' character
?	a single instance of the '?' character
	a single instance of the ' ' character
\(and \)	used for grouping
\{ and \}	used for expressing ranges of instances

In *extended* regular expressions:

Character Meaning

+	one or more instance
?	zero or one instance
	either the left or the right hand alternative
(and)	used for grouping
{ and }	used for expressing ranges of instances

In extended regular expressions, if you wish to use a literal +, |, (,), {, or } character in your regex, you must escape it with a \. For example:

The *basic* validation regex `^(hi)\{2,5\}` is written as `^(hi){2,5}` after the conversion to extended regular expressions.

A basic regex like `you+me` must now be expressed as `you\+me` because + means *one or more*. Therefore, the extended regex `you+me` matches strings like `youme`, `youume`, `youuume`, etc.

You should probably revisit your validation regexes, since the extended regular expressions now being used allow for stricter input checking.



DataDeploy Database Auto-Synchronization

This appendix describes how to configure and use the database auto-synchronization (DAS) module.

Overview

The DAS module is bundled with DataDeploy. After you configure DAS, it automatically deploys data content records (DCRs) to a database whenever a TeamXpress user performs any of the following actions:

- Creates, changes, or deletes a DCR through the TeamXpress templating GUI.
- Creates, changes, or deletes a file, TeamXpress area, or branch containing extended attributes via the command line.
- Creates, changes, or deletes a file, TeamXpress area, or branch containing extended attributes via the TeamXpress file system interface.

DAS accomplishes this by running DataDeploy as a daemon, and by using various TeamXpress events as triggers to initiate deployment. The following sections describe how to configure and run DAS.

Software Requirements

To use DAS, you must first install and configure the following Interwoven products:

- TeamXpress (see the *TeamXpress Administration Guide*)
- TeamXpress Templating
- DataDeploy

DAS Program and Configuration Files

The following files control the operation of DAS. All but the last file, `iw.cfg`, are installed automatically when you install DataDeploy (`iw.cfg` is installed automatically with TeamXpress). See the sections following the table for configuration instructions.

File	Location	Description
<code>daemon.cfg</code>	<code>dd-home/conf</code>	Configuration file used by the DataDeploy daemon for start-up. You do not need to configure <code>daemon.cfg</code> before running DAS. However, you can optionally add <code><allowed-hosts></code> and <code><bind></code> tags to <code>daemon.cfg</code> to further control access to the database server. See Item 16 in “Sample File Notes” for more information.
<code>ddcfg.template</code>	<code>dd-home/conf</code>	Template DataDeploy configuration file used by <code>ddgen.ipl</code> as a basis for creating all the working DataDeploy configuration files for the templating data types. You must configure <code>ddcfg.template</code> as described in “Editing <code>ddcfg.template</code> and <code>drop.cfg</code> ” on page 385 before running DAS.
<code>ddgen.ipl</code>	<code>dd-home/bin</code>	DataDeploy configuration file generator. You do not need to configure <code>ddgen.ipl</code> before running DAS.
<code>drop.cfg</code>	<code>dd-home/conf</code>	Utility configuration file used by the DataDeploy daemon when dropping tables. You must configure <code>drop.cfg</code> as described in “Editing <code>ddcfg.template</code> and <code>drop.cfg</code> ” on page 385 before running DAS.
<code>iwsyncdb.cfg</code>	<code>dd-home/conf</code>	Configuration file for <code>iwsyncdb.ipl</code> . Controls name and port number for the DataDeploy daemon host. Also controls DataDeploy event logging. See “Editing <code>iwsyncdb.cfg</code> ” on page 386 for more information.
<code>iwsyncdb.ipl</code>	<code>dd-home/bin</code>	TeamXpress event trigger program and CLT. You do not need to configure <code>iwsyncdb.ipl</code> before running DAS.
<code>iw.cfg</code>	<code>/etc</code>	Controls whether file renaming, moving, and deletion will trigger deployment. See “Editing <code>iw.cfg</code> ” on page 387 for more information.

Configuring DAS

You must perform the following steps to configure DAS following a DataDeploy installation:

- Edit configuration files that are specific to DataDeploy.
- Edit the main TeamXpress configuration file, `iw.cfg`.
- Run the main DataDeploy configuration script, `iwsyncdb.ipl`.

The following sections describe these steps in detail.

Editing DataDeploy Configuration Files

This section describes how to configure the `ddcfg.template`, `drop.cfg`, and `iwsyncdb.ipl` files with your site-specific information.

Editing `ddcfg.template` and `drop.cfg`

You must set the following attributes in each `<database>` element in `ddcfg.template` and `drop.cfg`:

Attribute	Set to...
<code>db</code>	The host, port, and name of the destination database. Syntax is " <code>hostname:portnumber:dbname</code> ".
<code>user</code>	The user name that DataDeploy uses when logging into the database server.
<code>password</code>	The password for <code>user</code> . Note that any password named here is not encrypted, and can be read by anyone having access to <code>ddcfg.template</code> .

For example, the following settings configure `ddcfg.template` so that DataDeploy will connect to the database server as `marketing` (using the password `$al45`) and deploy data to the `marketingdb` database on port `1521` of the server `dbserver1`:

```
<database db = "dbserver1:1521:marketingdb"
  user = "marketing"
  password = "$al45"
```

You must configure these settings within each occurrence of the `<database>` element. For example, if the `<database>` element occurs four times in `ddcfg.template`, you must configure these settings identically in all four locations. The same requirement applies to `drop.cfg`. You must reconfigure these settings in both files whenever you change to a different database, user, or password.

Editing `iwsyncdb.cfg`

The `dd-home/conf/iwsyncdb.cfg` file controls the following DataDeploy parameters.

Parameter	Setting in <code>iwsyncdb.cfg</code>
DataDeploy daemon host port number (host name is set automatically to the local host name of the TeamXpress server).	Set <code>daemon_port=number</code> . For example, to set the port number to <code>3456</code> , enter <code>daemon_port=3456</code>
Logging of DataDeploy events.	Set <code>suppress_log=yes</code> to disable logging. Set to <code>no</code> to enable logging.
DataDeploy 3-tier or DAS operation.	Set <code>mode=das</code> to enable DAS mode. Set <code>mode=3tier</code> to enable 3-tier mode.

Editing iw.cfg

You must edit the [iwserver] section of `/etc/iw.cfg` as follows to support DAS recognition of TeamXpress events. Once configured, DAS will support these events whether they are initiated from the TeamXpress GUI, the TeamXpress file system interface, or the command line (via standard OS commands or TeamXpress command-line tools such as `iwextattr`).

TeamXpress Event	Setting in iw.cfg
Rename a file.	<code>log_renamefse=yes</code>
Move a file.	<code>log_renamefse=yes</code>
Delete a file.	<code>log_syncdestroy=yes</code>
Set extended attributes on a file.	<code>log_setea=no</code>
Delete extended attributes from a file.	<code>log_setea=no</code>
Revert a file containing extended attributes to an earlier version.	<code>log_syncrevert=yes</code>

Running `iwsyncdb.ipl`

This section describes how to run the `iwsyncdb.ipl` script, which performs the following activities:

- Generates DataDeploy configuration files for use by the DataDeploy daemon.
- Submits the generated DataDeploy configuration files to the staging area and publishes an edition based on the updated staging area.
- Establishes TeamXpress events as triggers for automatic data deployment.
- Starts the DataDeploy daemon.
- Creates initial base and delta tables in the destination database for the updated TeamXpress areas.

The following sections and diagrams explain these activities in detail.

Starting `iwsyncdb.ipl`

Enter the following command to start the `iwsyncdb.ipl` script:

```
dd-home/bin/iwsyncdb.ipl -initial workarea_vpath
```

For `workarea_vpath`, specify the full path to the TeamXpress Templating workarea that was set up earlier as described in Chapter 2, “Initial Configuration.” For example, you would enter the following if the TeamXpress Templating subbranch `b1` and workarea `w1` are on the `default/main` branch, and `dd-home` is `/usr/iw-home/datadeploy`:

```
/usr/iw-home/datadeploy/bin/iwsyncdb.ipl -initial /default/main/dev/b1/WORKAREA/  
w1
```

`iwsyncdb.ipl` Activities

The following figures show the activities that take place when `iwsyncdb.ipl` runs. Activities are grouped as follows:

- Figure 1: Generation of DataDeploy Configuration Files
- Figure 2: Other DAS Setup Activities

All of the activities shown in Figures 1 and 2 take place when you enter `iwsyncdb.ipl` on the command line. You do not need to enter `iwsyncdb.ipl` a second time to initiate the activities shown in Figure 2.

Generation of DataDeploy Configuration Files

The following figure shows how DataDeploy configuration files are generated, submitted, and published when the `iwsyncdb.ipl` script runs. See the diagram key following the diagram for details about each step.

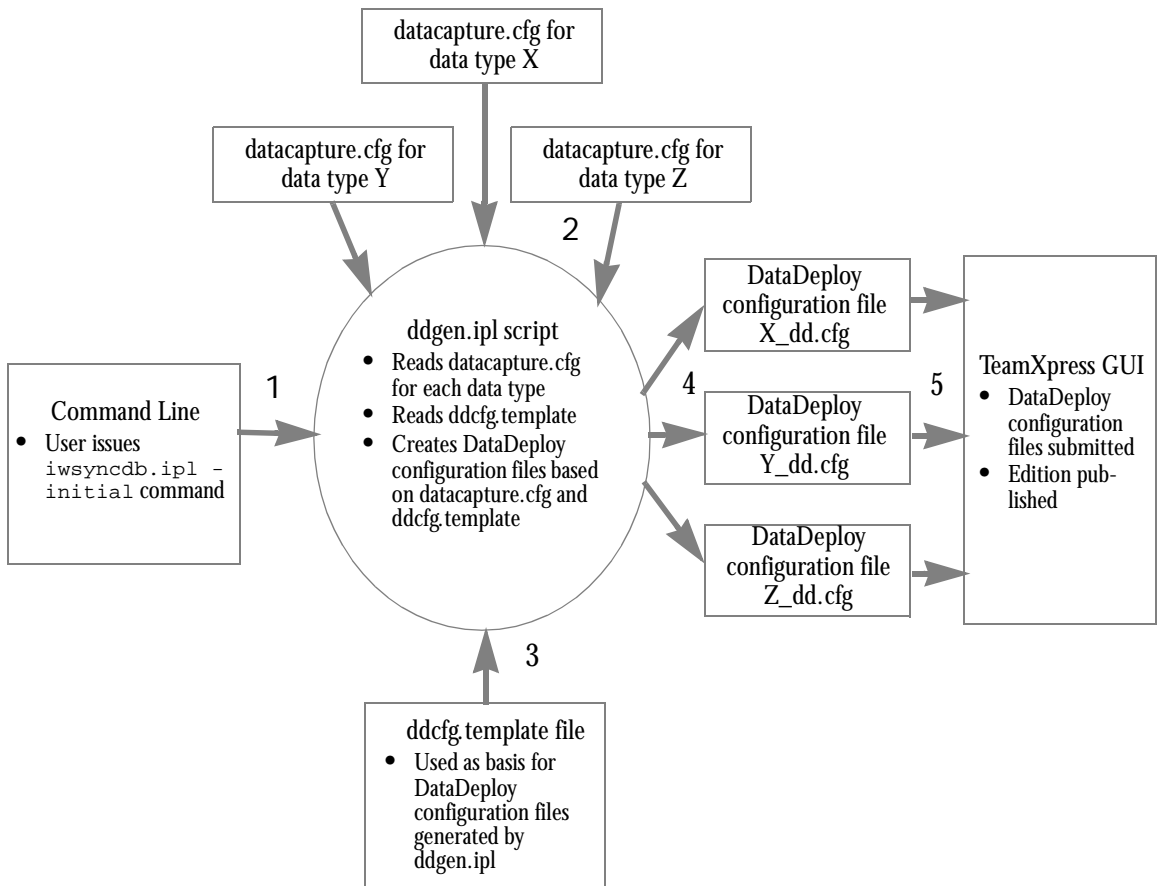


Figure 1: Generation of DataDeploy Configuration Files

Figure 1 Diagram Key

1. The `iwsyncdb.ipl -initial` command is executed from the command line as described in “Starting `iwsyncdb.ipl`” on page 388. The `iwsyncdb.ipl` script starts the `ddgen.ipl` script.
2. The `ddgen.ipl` script reads the TeamXpress `datacapture.cfg` file for each data type that exists in `workarea_vpath` specified in Step 1. For example, if the TeamXpress templating directory structure in `workarea_vpath` contains the data types `x`, `y`, and `z`, the `datacapture.cfg` file for each is read by `ddgen.ipl`.
3. The `ddgen.ipl` script uses `ddcfg.template` as the base format of the DataDeploy configuration files that it will generate for each data type.
4. Based on `ddcfg.template` and the `datacapture.cfg` files for each data type, `ddgen.ipl` creates DataDeploy configuration files for each data type. Continuing with the example from Step 2, the DataDeploy configuration files `x_dd.cfg`, `y_dd.cfg`, and `z_dd.cfg` are created. These configuration files configure a TeamXpress-to-database deployment similar to that described in “Sample TeamXpress-to-Database Configuration File” on page 175. The `mdc_dd.cfg` file is also created to ensure that DataDeploy remains synchronized with other TeamXpress features such as metadata capture and metadata search.
5. The newly generated DataDeploy configuration files are submitted to the staging area, and an edition based on the updated staging area is published.

Other DAS Setup Activities

The following figure shows how the remaining DAS setup activities take place when the `iwsyncdb.ipl` script runs. See the diagram key following the diagram for details about each step.

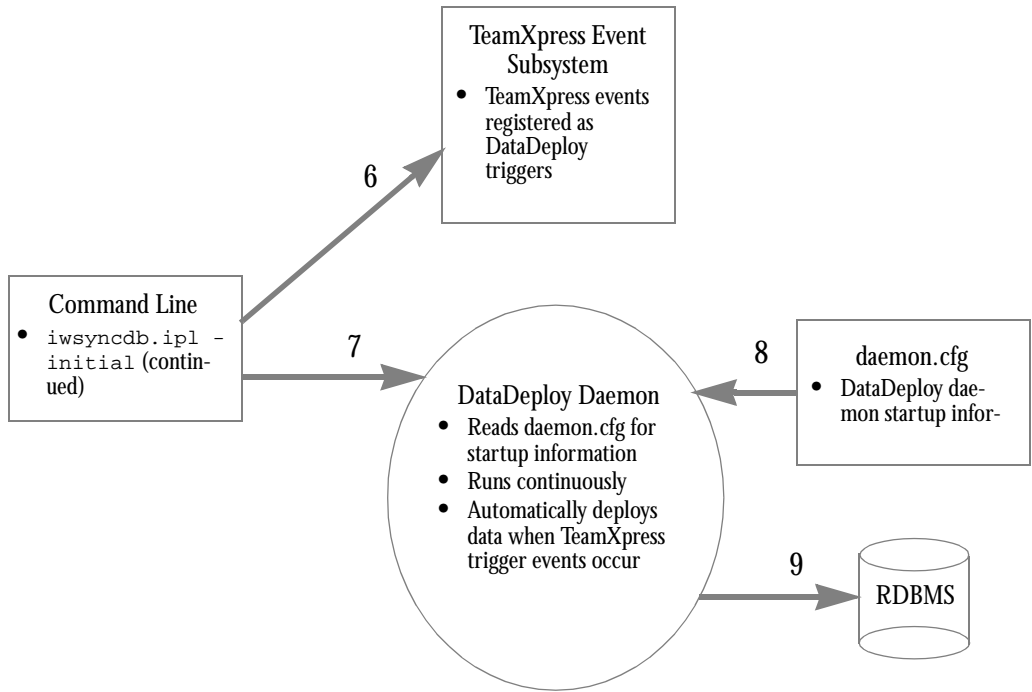


Figure 2: Other DAS Setup Activities

Figure 2 Diagram Key

- 6. The `iwsyncdb.ipl` script registers a default set of TeamXpress events as triggers that will automatically initiate deployment. See “TeamXpress Event Triggers” on page 396 for details about which events are registered as triggers.
- 7. The `iwsyncdb.ipl` script starts the DataDeploy daemon.



8. The DataDeploy daemon reads the `daemon.cfg` file, which contains additional daemon startup information. The daemon finishes its startup, and runs continuously until DAS is disabled as described in “Disabling DAS” on page 399.
9. The DataDeploy daemon creates the following in the destination database:
 - Initial wide base tables for the branch.
 - Initial delta tables and views for the workarea.

DAS is now configured and ready for use. The only time you need to repeat any configuration step is when you enable a different database, user, or password. If you add new templating branches, workareas, or files through the TeamXpress GUI, DAS automatically generates the necessary DataDeploy configuration files and initial tables.

Using DAS

After DAS is configured, it is transparent to TeamXpress templating end users. Therefore, there are no additional tasks that an end user must perform to use DAS. The following diagram shows how DAS automatically updates the necessary tables when a TeamXpress trigger event occurs. See the diagram key following the diagram for details about each step.

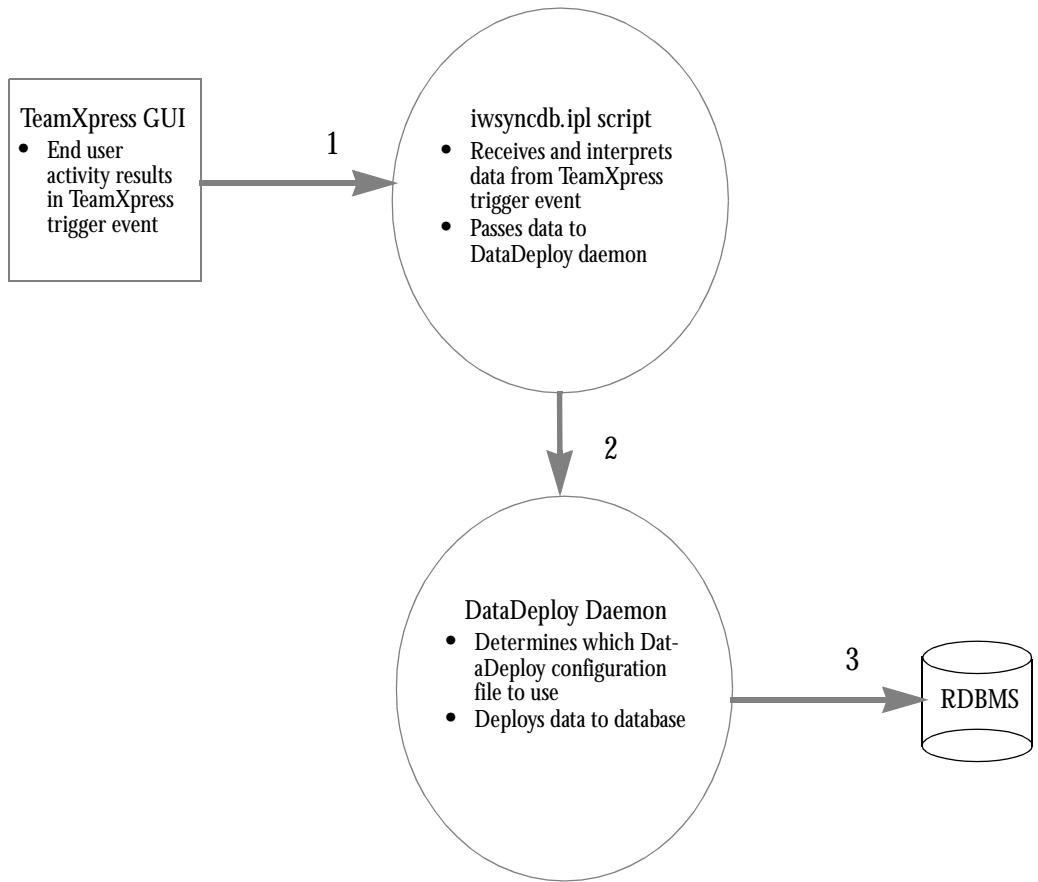


Figure 3: Using DAS

Figure 3 Diagram Key

1. TeamXpress templating end user activity (i.e., any activity shown in “TeamXpress Event Triggers” on page 396) results in a TeamXpress event trigger. The event trigger starts the `iwsyncdb.ipl` script and sends the changed data to the script.
2. The `iwsyncdb.ipl` script sends the DCR data to the DataDeploy daemon. The daemon determines which DataDeploy configuration file(s) to use for the deployment. For TeamXpress events (e.g., **Create Branch**) that are not specific to a single file, the daemon uses the `templating.cfg` file to determine which data types (and therefore which DataDeploy configuration files) are affected by the TeamXpress event. For example, in the case of a **Create Branch** TeamXpress event, the daemon reads `templating.cfg` to determine which data types exist in the branch. The daemon then uses the DataDeploy configuration files for each affected data type when deploying the new data to the database.

For events that are file-specific (e.g., renaming a file, etc.), the daemon uses the information from the TeamXpress event information module to determine which file is affected and which DataDeploy configuration file to use.

3. The daemon uses the appropriate DataDeploy configuration file(s) to update the affected base and delta tables in the database. The following section, “Table Update Details” describes these updates.

Table Update Details

This section describes how the base and delta tables described in the preceding section change as data is deployed. This example shows a hypothetical update to a data content record. In this example:

- Data category is `internet`.
- Data type is `pr` (press release).
- Branch is `b1`.
- Workarea is `w1`.

Table Naming Conventions

Base and delta tables use the following naming convention:

datacategory_datatype__branchname_areaname

Note the use of double underbars between *datacategory_datatype* and *branchname_areaname*. For example:

internet_pr__b1_staging (a base table for the staging area on the default/main/dev/b1 branch)

internet_pr__b1_workarea_w1 (a delta table for the workarea w1 on the default/main/dev/b1 branch)

Table Update Examples

When the initial wide base table is created as described earlier in Figure 2, Step 9, it contains a Path column, a State column, and columns for each *item* in the data content record. In this starting state, the table does not yet contain any values. Assuming that first three items are *PressDate*, *Headline*, and *Picture*:

Path	State	PressDate	Headline	Picture	...

Wide Base Table for Staging Area (Starting State)

When the initial delta table is created, it contains the same columns as the initial base table, plus values for each item:

Path	State	PressDate	Headline	Picture	...
f1	New	11/17/99	New Candidate Enters Race	cand.gif	...

Delta Table for Workarea (Starting State)

When the data content record is saved, its delta table values are transferred to the base table, and its own cells are cleared:

Path	State	PressDate	Headline	Picture	...
f1	New	11/17/99	New Candidate Enters Race	cand.gif	...

Base Table for Staging Area (Ending State)

Path	State	PressDate	Headline	Picture	...

Delta Table for Workarea (Ending State)

TeamXpress Event Triggers

DAS interprets the following TeamXpress events as deployment triggers. The event can be initiated from the TeamXpress GUI, the TeamXpress file system interface, or the command line. Whenever one of these events occurs, the delta and base tables are updated as shown here.

TeamXpress Event	Delta Table Action	Base Table Action
Create Branch	None.	Build empty base tables.
Create Workarea	Build delta tables.	None.
Delete Branch	Drop delta tables.	Drop base tables.
Delete Workarea	Drop delta tables.	None.
Modify DCR	Update or insert a new row.	None.
Add DCR	Insert a new row.	None.
Delete DCR	Insert or update the Not Present row.	None.

TeamXpress Event	Delta Table Action	Base Table Action
Submit modified DCR	<ol style="list-style-type: none"> 1. The Previous Staging row is propagated to all workareas except the submitting workarea. 2. Delete Previous Staging row from submitting workarea. 	Update the Staging row.
Submit added DCR	<ol style="list-style-type: none"> 1. The Placeholder row marked NOT-PRESENT is propagated to all workareas except the submitting workarea. 2. Delete the Placeholder row from the submitting workarea. 	Update the Staging row.
Submit deleted DCR	<ol style="list-style-type: none"> 1. The previous Staging row is propagated to all workareas except the submitting workarea. 2. Delete the previous Staging row from the submitting workarea. 	Update the Staging row.
Get Latest (workarea)	Rebuild the delta tables.	None.
Copy To (any area)	Rebuild the delta tables.	None.
Rename Workarea	<ol style="list-style-type: none"> 1. Drop the old delta tables. 2. Regenerate new delta tables. 	None.
Rename Branch	None.	<ol style="list-style-type: none"> 1. Drop the old base tables. 2. Regenerate new base tables.
Rename Directory	Regenerate new delta tables.	None.
Rename File	<ol style="list-style-type: none"> 1. Delete the row for the old file name. 2. Add a row for the new file name. 	None.
Move File	<ol style="list-style-type: none"> 1. Delete the row for the old file name. 2. Add a row for the new file name. 	None.

TeamXpress Event	Delta Table Action	Base Table Action
Delete File	If a row for the file exists in the base table, the row in the delta table is marked NOT-PRESENT. If no row existed in the base table, the row in the delta table is deleted.	None.
Set extended attributes	Insert or update the row.	None.
Delete extended attributes	In a wide table, rows are updated. In a narrow table, the row is deleted or marked NOT-PRESENT.	None.
Revert	Use the data from the earlier version of the file (selected in the TeamXpress GUI) to update or insert a new row.	None.

Logging DAS Activities

By default, all DAS activities are logged in `dd-home/iwevents.log`. If this logging has an adverse affect on system performance, you can optionally turn off logging for any of the TeamXpress events shown in the table in “Editing iw.cfg” on page 387. Use the following event names when disabling logging:

```
RenameFSE
SyncDestroy
SetEA
DeleteEA
SyncRevert
```

For example, to prevent **Rename** events from being logged, set the following in `iw.cfg`:

```
iwevents_exclude="RenameFSE"
```

You can also use regular expressions with the following syntax to further control event logging:

```
renamefse_filter="REGEX"
```


For example, to specify that only **Rename** events occurring in the workarea `bill` are logged:

```
[iwservr]
renamefse_filter="/default/main/dev/WORKAREA/bill "
```

This entry sets regular expressions, one of which must match the event line (as seen in `iwevents.log`) in order for an event to be logged. If these are empty or absent, all corresponding events are logged.

Disabling DAS

Issue the following command to remove the TeamXpress event trigger scripts and stop the DataDeploy daemon:

```
dd-home/bin/iwsyncdb.ipl -uninstall
```

To re-enable DAS after it has been disabled, issue the following command:

```
dd-home/bin/iwsyncdb.ipl -install
```

Note that you do not need to regenerate the `datacapture.cfg` files that were generated earlier during DAS configuration. See the next section, “`iwsyncdb.ipl Usage`” for more information about the `iwsyncdb.ipl` command.

`iwsyncdb.ipl Usage`

Usage

```
iwsyncdb.ipl [
    -h | -install | -uninstall | -iwat | -iwrmat |
    -startddd | -stopddd | -ddgen vpath [dcr-type] [-force] |
    -initial vpath [dcr-type] | -mcdggen [-force] |
    -resyncbr vpath [dcr-type] | -resyncwa vpath [dcr-type] |
    -rmbr vpath [dcr-type] | -rmwa vpath [dcr-type] |
    -showbase vpath [dcr-type] | -showdelta vpath [dcr-type] |
    -showtracker | -synctracker vpath
```



-]
- install Installs the database synchronization triggers and starts the DataDeploy daemon.
 - uninstall Removes the TeamXpress event trigger scripts and stops the DataDeploy daemon.
 - iwat Registers the `iwsyncdb` trigger scripts.
 - iwrmat Unregisters the `iwsyncdb` trigger scripts.
 - startddd Starts the DataDeploy daemon.
 - stopddd Stops the DataDeploy daemon.
 - ddgen *vpath* [*dcr-type*] Generates DataDeploy configuration files for data types configured in `templatedata/templating.cfg` under the specified workarea *vpath*. The `-force` option overwrites any existing configuration files. The optional *dcr-type* setting specifies a single data type (rather than all data types in *vpath*) to generate a configuration file for. If used, *dcr-type* must be the last argument in the option list.
 - initial *vpath* [*dcr-type*] Generates the initial base and delta tables for the first template-enabled workarea *vpath*. The optional *dcr-type* setting specifies a single data type (rather than all data types in *vpath*) to generate tables for.
 - mdcddgen Generates the DataDeploy configuration file `mdc_dd.cfg` (based on `iw-home/local/config/datacapture.cfg`) for use by the metadata capture subsystem. The `-force` option overwrites any existing configuration files.
 - resyncbr *vpath* [*dcr-type*] Regenerates the base tables for the branch named by *vpath* and the workareas for the underlying workareas. The optional *dcr-type* setting specifies a single data type (rather than all data types in *vpath*)

- to resync tables for. You should run `iwfreeze` to freeze the backing store before regenerating.
- `-resyncwa vpath [dcr-type]` Regenerates the delta tables for the workarea `vpath`. The optional `dcr-type` setting specifies a single data type (rather than all data types in `vpath`) to resync tables for. You should run `iwfreeze` to freeze the backing store before regenerating.
- `-rnbr vpath [dcr-type]` Destroys the base tables for the branch named by `vpath`. The optional `dcr-type` setting specifies a single data type (rather than all data types in `vpath`) to destroy tables for.
- `-rmwa vpath [dcr-type]` Destroys the delta tables for the workarea named by `vpath`. The optional `dcr-type` setting specifies a single data type (rather than all data types in `vpath`) to destroy tables for.
- `-showbase vpath [dcr-type]` Shows the base table of the DCR type for the specified base path (e.g., `/default/main/dev/br/STAGING`). The optional `dcr-type` setting specifies a single data type to display.
- `-showdelta vpath [dcr-type]` Shows the delta table of the DCR for the specified workarea path (e.g., `/default/main/dev/br/WORKAREA/wa`). The optional `dcr-type` setting specifies a single data type to display.
- `-showtracker` Shows the tracker table containing all registered tables deployed via DataDeploy.



DataDeploy Database Server Configuration

Overview

This appendix describes the database server configuration tasks you must perform to configure the following databases to work with DataDeploy:

- IBM DB2 UDB 6.1
- Sybase ASE 11.5
- Informix 7.3

IBM DB2

DataDeploy supports IBM DB2 UDB 6.1 on Windows NT/2000 systems. The following sections describe how to configure the database server to work with DataDeploy.

Setting Page and Table Sizes

The default pagesize for a tablespace in DB2 is 16K, which is too small for the examples shipped with TeamXpress Templating (the examples require that a tablespace of pagesize 32K be already set up on the DB2 server). Also, the default column size and datatype used by DataDeploy is `VARCHAR (300)`. These conditions require that you perform one of the following procedures:

1. Make sure that the default tablespace matches the required pagesize (32K). The default tablespace is usually named `IBMDEFAULTGROUP`. Or:
2. Create a tablespace with the required pagesize (32K) and specify the tablespace name as follows in the `<database>` element in the DataDeploy configuration file:

```
<database db = "//host:port/database"  
  user = "username"  
  password = "password"
```

```
table = "tablename"  
vendor = "ibm"  
tablespace = "tablespacename"  
max-id-length = "30">
```

The `tablespace` attribute is valid only for DB2 configuration. It is ignored if you set it when using any other database.

Installing and Starting JDBC

DB2's JDBC driver (`db2java.zip`) is installed with other JDBC drivers (Oracle, Sybase ASE, and Informix). The driver class `COM.ibm.db2.jdbc.net.DB2Driver` that DataDeploy uses to connect to a DB2 database requires that the DB2 client is also installed. See the documentation supplied by the database vendor for information about installing the DB2 client.

DB2 does not start the daemon to accept JDBC connections by default. You must do this manually by executing the following command:

```
db2jstrt port
```

The `port` number you enter on the command line must match the `port` number shown in the `db` attribute in "Setting Page and Table Sizes" on page 403. If you do not specify a value for `port`, it takes a default value of 6789.

Sybase ASE

DataDeploy supports Sybase ASE 11.5 on Windows NT, Windows 2000, and Solaris systems. The following sections describe how to configure the database server to work with DataDeploy.

Enabling DDL Statements

You must enable DDL statements for transactions as follows. Note that this cannot be done for the master db.

```
1> sp_dboption dbname, "ddl in tran", true
```

Setting Sort Order

Set up case-insensitive sort order for the database by executing the `$SYBASE/bin/sqlloc` utility to set case-insensitive dictionary order. You will also need to recreate indexes on the database that was changed, unless the sort order was changed on initial installation.

Install Stored Procedures

Install jconnect 4.2 stored procedures as follows:

1. Download the jConnect 4.2 package from the Sybase website.
2. Follow the instructions in the “Sybase jConnect for JDBC Installation Guide,” Chapter 1, section “Adaptive Server Enterprise” to install the stored procedures for JDBC support into the database.

Informix

DataDeploy supports Informix 7.3 on Windows NT systems, Windows 2000, and Solaris systems. The following sections describe how to configure the database server to work with DataDeploy.

Enabling Logging

Any databases created for use with Informix must be created with logging enabled. This can be accomplished with the Informix tool `dbaccess`, using an SQL command such as the following:

```
create database xyzdb with log
```



Appendix E

DataDeploy Querying Tables

This appendix describes how to query tables through SQL commands that you execute manually after deployment. Methodology differs depending on table type.

Note: You can also embed SQL commands in the DataDeploy configuration file's `<sql>` element. These commands execute automatically during deployment and do not require you to manually query the database. See “Invoking DataDeploy” on page 207 for more information.

Querying Base and Standalone Tables

You can use simple SQL statements specifying key-value pair criteria when querying a base or standalone table. For example:

```
SELECT path FROM staging
    WHERE key = News-Section AND value = Sports;
```

Querying Delta Tables

To query a delta table, you can first create a view consisting of a complex query and then apply a simple query on the view. For example:

```
CREATE VIEW areaview ( key, value, path ) AS
    SELECT key, value, path
    FROM sa
    WHERE NOT EXISTS
        ( SELECT *
          FROM wa_x WHERE
            wa_x.key = sa.key AND
            wa_x.path = sa.path )
    UNION
    SELECT key, value, path
    FROM wa_x WHERE wa_x.state != 'NotPresent';

SELECT path FROM areaview
    WHERE key = News-Section AND value = Sports
```

The `CREATE VIEW` command in this example is the default DataDeploy schema that executes when `table-view` is set to `yes` in the DataDeploy configuration file's `<database>` element.

OpenDeploy Client and Server Configuration File Options

Many OpenDeploy configuration options may be specified in either the client or server configuration files. Some must be specified in both. However, some configuration options are specific to client or server configuration files. The following table lists all available configuration file options, and whether they are specified on the client or the server.

Configuration File Option	Configuration File	Page
Specifying Connections and Locations		
<code>allowed_directory=</code> <i>path</i>	server	page 290
<code>port=#</code>	server	page 289
<code>TeamSite_server=</code> <i>name</i>	server	page 289
<code>remote_directory=</code> <i>absolute_path</i>	client	page 263
<code>remote_port=#</code>	client	page 262
<code>remote_server=</code> <i>server</i>	client	page 262
Specifying Timeouts		
<code>timeout=#</code> <i>seconds</i>	either	pages 263 and 291
Security Options		
<code>client_is_trusted=yes no</code>	server	page 290
Specifying Deployment Sections		
<code>deployment=</code> <i>name</i>	client or both	pages 262 and 290
Specifying Locations of Files to Be Deployed		
<code>area=</code> <i>path</i>	client	page 264
<code>hostname=</code> <i>name</i>	client	page 263



Configuration File Option	Configuration File	Page
<code>local_directory=path</code>	client	page 264
Specifying Which Files to Deploy		
<code>date_different</code>	client	page 268
<code>file_list=path</code>	client	page 273
<code>previous_area=path</code>	client	page 271
<code>revert</code>	client	page 267
<code>TeamSite_based</code>	client	page 270
Specifying Which Files to Exclude		
<code>destination_exclude=path</code>	either	pages 278 and 292
<code>destination_exclude_pattern=pattern</code>	either	pages 279 and 292
<code>exclude=path</code>	client	page 280
<code>exclude_pattern=pattern</code>	client	page 280
<code>source_exclude=path</code>	client	page 276
<code>source_exclude_pattern=pattern</code>	client	page 277
Renaming and Deleting Files During Deployment		
<code>do_deletes</code>	either	pages 280 and 293
<code>rename_suffix=suffix</code>	either	pages 280 and 293
Changing Permissions on Files During Deployment		
<code>amask=mask</code>	either	pages 281 and 293
<code>changeaccess=ACL</code>	either	pages 283 and 295
<code>dir_perm=permission</code>	either	pages 281 and 293

Configuration File Option	Configuration File	Page
<code>file_perm=<i>permission</i></code>	either	pages 281 and 294
<code>group=<i>groupid</i></code>	either	pages 282 and 294
<code>group_translations</code>	client	page 282
<code>ignore_groups</code>	either	pages 282 and 294
<code>ignore_modes</code>	either	pages 282 and 294
<code>ignore_users</code>	either	pages 282 and 294
<code>omask=<i>mask</i></code>	either	pages 282 and 294
<code>setaccess=<i>ACL</i></code>	either	pages 283 and 295
<code>user=<i>userid</i></code>	either	pages 283 and 294
<code>user_translations</code>	client	page 283
Encryption		
<code>key_file=<i>path</i></code>	both	pages 285 and 296
<code>ssl_certificate=<i>path</i></code>	either	pages 285 and 297
<code>ssl_ciphers=<i>ciphers</i></code>	either	pages 285 and 297
<code>ssl_privatekey=<i>path</i></code>	either	pages 285 and 297
Deploy and Run		
<code>as=<i>username</i></code>	either	pages 286 and 298



Configuration File Option	Configuration File	Page
<code>async=yes</code>	either	pages 287 and 299
<code>deploy_run_script=script_to_run</code>	either	pages 286 and 298
<code>dir_mask=dir</code>	either	pages 287 and 299
<code>disable_scripts=yes</code>	server	page 297
<code>file_mask=file</code>	either	pages 287 and 299
<code>require_abs_script_path=yes</code>	server	page 288
<code>when=condition</code>	either	pages 286 and 298
<code>where=dir</code>	either	pages 288 and 300
Links		
<code>destination_follow_links</code>	client	page 288
<code>follow_links</code>	client	page 288
<code>source_follow_links</code>	client	page 288
Debugging Deployment Configuration		
<code>dont_do</code>	client	page 288
Authentication by IP Address		
<code>allowed_hosts=hostlist</code>	server	page 300
<code>bind_address=address</code>	server	page 300

Index

- A**
 - ACLs 284, 295
 - evaluating 368
 - adding replicants 50
 - advanced features
 - authentication by IP
 - address 301
 - Deploy and Run 313
 - encryption 305
 - allowed element 47, 53, 67, 133, 137
 - allowed users 258
 - and element 47, 67, 133, 137
 - architecture
 - three-tier 145, 147
 - two-tier 145, 146
 - asynchronous mode 315, 319
 - author_submit_dcr.wft 141
 - authorization configuration
 - file 258
 - auto-synchronization, database
 - see* database
 - available_templates.ipl 30, 140
 - editing 31
- B**
 - base tables 212
 - generation 163
 - naming conventions 395
 - querying 407
 - updating 166, 394
 - examples 395
 - boolean tests 83
 - branch element 134, 136
 - browser element 46, 66
- C**
 - callout element 47, 54
 - category 22
 - category element 132, 136
 - checkbox element 48, 66
 - ciphers 311
 - client configuration files 251, 253, 328, 331, 335, 343
 - client configuration options 261, 310, 409
 - client options
 - changing file permissions 281
 - debugging 288
 - deleting files 280
 - Deploy and Run 286
 - deployment sections 262
 - deployment targets 262
 - deployment timeouts 263
 - encryption 285
 - excluding files 275
 - overview 261
 - renaming files 280
 - specifying files to be
 - deployed 263
 - symbolic links 288
 - client versus server 258
 - clients
 - trusted 258
 - CLT
 - iwductacleval 368
 - iwtdtd2sym 354, 370
 - iwgen 371
 - iwpt_compile.ipl 372
 - iwregen 375
 - iwsym2dct 358, 376
 - iwxml_validate.ipl 377
 - upgrade_dct.cfg.ipl 379
 - command line options
 - client 240
 - general 239
 - server 240
 - comparison
 - directory 265
 - file lists 273
 - reverting files 267
 - component directory 23
 - component template 71
 - example 73
 - configuration file
 - authorization 258
 - configuration files 152, 251
 - available_templates.ipl 30, 31
 - client 251, 253, 328, 331, 335, 343
 - debugging 288
 - deployment sections 253, 254

- local directory
 - sections 253, 254
 - scope of options 255
- components 153
- daemon.cfg 384
- database auto-
 - synchronization 384
 - editing 385
- database.xml 213
 - editing 227
- database-to-database 197
- database-to-XML 198
- datacapture.cfg 20, 22
 - example 41, 59
- DataDeploy 140, 212
 - location 227
- ddcfg template 384
 - editing 385
- ddsync.ipl 213, 220
- DNR scripts 217, 218
- drop.cfg 384
 - editing 385
- elements
 - client section 182
 - columns to update 193
 - database section 187
 - Database-to-Database 173
 - Database-to-XML 173
 - deployment section 182
 - destination section 186
 - filter section 181
 - include file 181
 - required 171
 - rows to update 191
 - server section 195
 - source data location 184
 - source section 183
 - source type 183
 - SQL section 194
 - substitution section 181, 185
 - TeamXpress-to-
 - Database 172
 - TeamXpress-to-XML 172
 - update type and related data 193
 - XML-to-Database 174
 - XML-to-XML 174
 - encryption options 310
 - generated using iwsynch.ipl script 389
 - iw.cfg 385
 - editing 387
 - iwsynchdb.cfg 384
 - editing 386
 - loaddb.cfg 213
 - generating 226
 - oddd_receive.cfg 214, 217, 221
 - location 221
 - oddd_send.cfg 214, 217, 218, 221
 - OpenDeploy 212
 - parameter substitutions 175
 - presentation template 23
 - server 221, 251, 317, 327, 330, 335, 340
 - starting-state base table 204
 - subxmldb.template 213
 - synchronized deployment 213, 222
 - TeamXpress-to-database 175
 - TeamXpress-to-XML 196
 - templating cfg 20, 25, 30, 129
 - example 130
 - tsxml.cfg 214, 217
 - editing 224
 - workflow 30
 - XML-to-database 200
 - XML-to-XML 202
 - configuration options 258, 409
 - content
 - conditional inclusion 91
 - creating 25
 - creating records 18
 - conventions
 - notation 6
 - path name 7
 - coordinating server and client
 - configuration files 254
 - cred element 47, 67, 133, 137

D

 - data
 - sizes 157
 - types 157
 - data capture form 40
 - example 57
 - data capture subsystem 18, 19
 - data capture symbol table
 - creating 370
 - transforming 376
 - data capture template
 - creating 376
 - creating from DTDs 353
 - customizing 38
 - definition 18
 - DTD 64
 - example 39
 - overview 38
 - data category 22
 - making available 132
 - data content record
 - creating 25
 - definition 19, 23
 - example 54, 57, 73
 - initiating workflow 31

- searching 34
- data directory 22
- data type 22
 - making available 132
- database
 - auto-synchronization 146, 147, 383
 - configuration files 384
 - configuring 385
 - disabling 399
 - event triggers 396
 - logging 398
 - overview 383
 - programs 384
 - software 383
 - usage 392
 - object name lengths 156
 - servers
 - configuration 403
 - IBM DB2 403, 404
 - Informix 405
 - Sybase ASE 404
 - database element 44
 - datacapture.cfg 20, 22, 25, 38, 41, 353
 - creating 358
 - example 59
 - data-capture-requirements
 - element 43, 64
- DataDeploy
 - configuration files 140
 - daemon 147
 - integrated with TeamXpress
 - Templating 140
 - invoking 207
 - running as a workflow job 139
 - service 147
 - running 210
 - synchronization
 - deployment 211
 - data-type element 132, 136
 - date-different option 268
 - ddgen.ipl command 384
 - ddsynch.ipl command 220
 - logging 221
 - syntax 220
 - usage 220
 - debugging 127
 - debugging tags 372
 - default element 67
 - deleting files 280, 292
 - deleting replicants 50
 - delta tables
 - generating 165
 - naming conventions 395
 - querying 407
 - updating 394
 - examples 395
 - Deploy and Run 297, 313
 - asynchronous mode 315
 - configuring the client 313
 - configuring the server 317
 - disabling 317
 - log files 318
 - scripts
 - asynchronous mode 319
 - output 320
 - specifying 313
 - deployment
 - automated 145, 146
 - comparison 265
 - configuration files 152
 - configuring 145, 251
 - database
 - base table 163, 166
 - data sources 159
 - data synchronization 163
 - delta table generation 165
 - destinations 159
 - details 163
 - overview 158
 - tables 168, 169
 - tuples 159
 - database auto-synchronization
 - triggers 396
 - executing 145
 - forward 326
 - incremental 157
 - installing 232, 233
 - invoking, methods 151
 - of different directories 333
 - overview 325
 - reverse 337
 - scenarios 158
 - security
 - encryption 305
 - server authentication 301
 - Site Rollback 346
 - synchronization 211
 - synchronized
 - base tables 212
 - configuration 212, 222
 - configuration files 213
 - differential 228
 - file interaction 222
 - full 228
 - invoking 227
 - OpenDeploy 217, 221
 - process 214
 - programs 213
 - software 213
 - TeamXpress
 - templates 211
 - syntax 239
 - through firewalls 301, 349

- to multiple servers 329
- transactional 242

- deployment directories
 - specifying 262

- deployment options
 - global 255
 - local directory 255
 - scope of 255

- deployment sections
 - multiple 262
 - specifying 262

- deployment targets
 - specifying 262

- description element 44

- directory comparison 268

- directory element 134, 136

- directory structure

- contents 22
 - copying 30
 - overview 21
 - sample 28

DTD

- converting to data capture templates 353

- data capture 63

- sample 354

- unsupported features 360

E

- EDITION, paths ending in 264

- editions

- specifying 263

- element

- allowed 47, 53, 67, 133, 137

- and 47, 67, 133, 137

- branch 134, 136

- browser 46, 66

- callout 47, 54

- category 132, 136

- checkbox 48, 66

- cred 47, 67, 133, 137

- data-capture-requirements 43, 64

- data-type 132, 136

- default 67

- description 44

- directory 134, 136

- inline 48, 53, 60

- item 44, 65

- locations 136

- not 47, 67, 133, 137

- option 48, 67

- or 47, 67, 133, 137

- presentation 133, 136

- radio 49, 66

- replicant 50, 67

- ruleset 43, 65

- select 51, 66

- template 134, 136

- templating 132, 136

- text 51, 65

- textarea 52, 66

- encryption 285, 296

- asymmetric 306

- ciphers 311

- configuring for

- asymmetric 310

- creating key files and

- certificates 306

- key files 285, 296, 305

- SSL 285, 297, 306

- symmetric 305

- errata 7

- evaluating ACLs 368

- event triggers 396

- example templating

- environment 28, 29

- copying 30

- excluding files 275, 291

F

- file lists 272

- file permissions

- changing 281, 293

- UNIX 281, 293

- Windows NT 283, 295

- files 41

- available_templates.ipl 30

- datacapture.cfg 20, 38, 59, 353

- deleting 280, 292

- DTD 63, 354

- excluding 275

- iw.cfg 34

- renaming 280, 292

- sample output from

- iwtdtd2sym CLT 354

- templating.cfg 20

- files to be deployed

- specifying 263

- firewalls 301, 349

- forward deployment

- of different directories 333

- to a single server 326

- to multiple servers 329

G

- generated HTML files 18, 26

- specifying locations 134

- Global Report Center 249

H

- handling replicated data 99

- hosts

- limiting access 304
 - specifying 263
- HTML pages
 - from presentation template
 - compiler 372
 - generating 26, 371
 - regenerating 375

I

- IBM DB2
 - configuration 403
 - JDBC
 - installing 404
 - starting 404
 - page setup 403
 - table size 403
- incremental deployment 157
- Informix
 - configuration 405
 - logging, enabling 405
- inline element 48, 53, 60
- installation
 - drivers 149
 - on Solaris 11
 - on Windows NT 13
 - Solaris 149
 - supported platforms 148
 - Windows NT 149
- installing OpenDeploy
 - UNIX 232
 - Windows NT 235
- instance
 - defined 38
- integrating 31
- interaction 222
- invoking deployment
 - UNIX 235
 - Windows NT 237

- IP address
 - authentication 301
- item element 44, 65
 - defined 38
- iw.cfg 34
- iw_case tag 83
- iw_else tag 88
- iw_if tag 89
- iw_ifcase tag 91
- iw_include tag 95
- iw_iterate tag 99
- iw_last tag 104
- iw_next tag 105
- iw_perl tag 106
- IW_PREV, paths ending in 264
- iw_pt tag 110
- iw_repeat tag 111
- iw_sql_iterate tag 112
- iw_sql_open tag 115
- iw_sql_query 118
- iw_sql_query tag 118
- iw_system tag 120
- iw_then tag 121
- iw_value tag 122
- iw_xml tags 82
- iwdd.ipl command 207
 - examples 209
 - syntax 207
 - usage 207
- iwdeploy syntax
 - client 239
 - server 239
- iwtd2sym 354, 370
- iwevents 272
- iwgen 371
- iwpt_compile.ipl 372
- iwregen 375

- iwsym2dct 358, 376
- iwsynch.ipl command 389
- iwsynchdb.ipl command 384
 - activities 388
 - running 387
 - starting 388
 - usage 399
- iwxml_validate 377

J

- Java callout 54
- JDBC
 - IBM DB2
 - installing 404
 - starting 404

K

- key files 305

L

- links 288
- local directories 263
- locations element 136
- logs 232, 233, 243
 - differences between UNIX and Windows NT 248, 319
 - in Deploy and Run 318
 - locations 247
 - names 247
 - verbose levels 243

N

- name lengths, database
 - objects 156
- narrow tuples *see* tuples
- not element 47, 67, 133, 137
- notation conventions 6
- notification of deployment 313

O

OpenDeploy

- database system assets 211
- file system assets 211
- server daemon, starting 222
- synchronization
 - deployment 211
 - overview 211
- synchronized deployment
 - configuration 217
 - modes, supported 221
- option element 48, 67
- options
 - configuration 258
- or element 47, 67, 133, 137
- output
 - inserting 120

P

- page generation subsystem 18,
19, 27
- parameter substitutions 175
- path name conventions 7
- paths
 - specifying 241
- platforms
 - Solaris 148
 - supported 148
 - Windows 2000 148
 - Windows NT 148
- port number
 - specifying 262
- presentation directory 23
- presentation element 133, 136
- presentation template 23, 99
 - adding Perl code 106
 - compiler 19, 71, 372

conditional inclusion of

- contents 121
- definition 19
- evaluating expression 89
- example 73
- guidelines 73
- including contents 88
- inserting a value 122
- inserting component
 - template 95
- inserting file 95
- iterating SQL result sets 112
- mapping 133
- naming 110
- opening database
 - connection 115
- querying a database 118
- repeating content 111
- skipping to last iteration of
 - loop 104
- skipping to next iteration of
 - loop 105
- preview directory 32, 33
- previewing data 135
- program
 - inserting output 120
- proxy server configuration 34

R

- radio element 49, 66
- regular expressions 277, 279, 280,
292
 - about 5
- renaming files 280, 292
- replicant element 50, 67
- replicants
 - adding 50
- reverse deployment 337

reverting files 267

- reverting websites 346
- ruleset element 43, 65
 - defined 38

S

scenarios

- deployment 158
- source/destination 145
- synchronization,
DataDeploy 211
- search menu item 34
- security
 - client versus server 289
- security options
 - client-side 290
 - server-side 290
- select element 51, 66
- server authentication 303
- server configuration files 327,
330, 335, 340
- server configuration options 310,
317
- server options
 - authentication by IP
 - address 300
 - changing file permissions 293
 - connections and locations 289
 - deleting files 292
 - Deploy and Run 297
 - deployment sections 290
 - deployment timeouts 291
 - encryption 296
 - excluding files 291
 - overview 289
 - renaming files 292
 - security 290
- servers

- IBM DB2 403
 - JDBC 404
 - page sizes 403
 - table sizes 403
 - Informix
 - configuration 405
 - logging, enabling 405
 - Sybase ASE 404
 - DDL statements 404
 - sort order 405
 - stored procedures 405
 - Site Rollback 346
 - Solaris, installation on 149
 - source/destination scenarios, supported 145
 - specifying
 - Deploy and Run scripts 313
 - paths 241
 - ports 289
 - server options 289
 - source servers 289
 - target directories 289
 - timeouts 263
 - SSL 306
 - standalone tables, querying 407
 - starting TeamXpress
 - Templating 34
 - Sybase ASE
 - configuration 404
 - DLL statements, enabling 404
 - sort order, setting 405
 - stored procedures, installing 405
 - synchronization,
 - OpenDeploy 211
 - synchronized deployment *see* deployment
 - syntax
 - Deploy and Run scripts 313–318
- T**
- tables
 - base 163, 166, 212
 - querying 407
 - updating 394
 - delta 165
 - querying 407
 - updating 394
 - naming conventions 395
 - querying 407
 - SQL 407
 - updating 168
 - examples 395
 - views, creating 169
- tags 118
- debugging 127, 372
 - iw_case 83
 - iw_else 88
 - iw_if 89
 - iw_ifcase 91
 - iw_include 95
 - iw_iterate 99
 - iw_last 104
 - iw_next 105
 - iw_perl 106
 - iw_pt 110
 - iw_repeat 111
 - iw_sql_iterate 112
 - iw_sql_open 115
 - iw_system 120
 - iw_then 121
 - iw_value 122
- target servers
 - specifying 262
 - template
 - component 71
 - template element 134, 136
 - templatedata directory 22
 - copying to workarea 30
 - templating directory
 - changing 32
 - templating element 132, 136
 - templating environment
 - example 29
 - templating cfg 20, 25, 30
 - customizing 129
 - DTD 136
 - example 130
 - text element 51, 65
 - textarea element 52, 66
 - three-tier architecture 145, 147
 - timeouts
 - specifying 263
 - transactional deployment 242
 - triggers, event 396
 - troubleshooting 127
 - trusted clients 258
 - tt_data 31
 - tt_deletedcr 32
 - tuples 159
 - defined 154
 - format 154
 - narrow 154, 161
 - wide 155, 162
 - two-tier architecture 145, 146
 - type 22
- U**
- uninstalling OpenDeploy
 - UNIX 234
 - Windows NT 237
 - upgrade_dct_cfg 379
 - user interface

- setting 33
- users
 - allowed 258

V

- validating XML 377
- validation regexes 44, 51
 - identifying 33
 - upgrading 379
- vpaths 254

W

- websites
 - reverting 346
- wide tuples *see* tuples
- Windows NT, installation on 149
- workflow 31
 - copying files for 30
 - DataDeploy process 140
 - initiating 32, 140
 - integrating with TeamXpress
 - Templating 140
 - preconfigured 141
 - schematic of 24, 26, 389

X

- XML 318
 - validating 377
- XML DTD 320
- XML log files
 - parsing 322