

Base Types Overview

Table of contents

| | |
|-----------------------------|---|
| 1 Intent..... | 2 |
| 2 Discussion..... | 2 |
| 3 Base Types..... | 3 |
| 4 Example..... | 6 |
| 5 Documentation Format..... | 6 |

FIXME (DRAFT):

Document Status: Draft.

1. Intent

We want a continuous, repeatable, transparent, and fully automated process that ties together the end-to-end build and deployment procedures for any application or stack of applications. We also want to ensure that this automation is reusable in any environment.

Working in Workbench, users utilize the Base Types to provide a set of basic abstractions that are assembled into a model that drives and coordinates the automation of the full application lifecycle -- from build to deployment and on through maintenance.

This document gives a brief overview to the collection of Base Types. Each base type represents a design pattern to managing an aspect of the build-to-deployment process. Users define their own patterns of automation by combining these base types.

2. Discussion

Problem

Traditionally, build and deployment activities follow a somewhat informal and mostly manual "recipe" — a grab bag of scripts, configurations, read-me files, emails, and recollections of random conversations. Developers and administrators responsible for implementing those procedures have to process volumes of complex information, hand-edit scripts and configurations, and then, through trial and error, get the release properly deployed and configured.

Even if there is partial automation, it ends up having to be retooled for each environment and most subsequent releases. Not only is this "recipe" process highly error prone and unpredictable, but you are basically paying for the same inefficient work over and over again, eating up your key staff's time.

Solution

The way to avoid these problems is to fully automate your build and deployment processes using an automation pattern. ControlTier enables you to quickly assemble an automation pattern for your applications by utilizing ControlTier's built-in Base Types. The ControlTier framework understands how to process that pattern and execute the commands found within it.

How much information do you need to capture in the pattern? Just enough so that you have a

representation of the architecture, configuration, and procedural knowledge needed to build and deploy your application. You do this by beginning with the Base Types which provide standard process building blocks, and model structure that represent both the procedures of the build and deployment process but also the structure of how your application or stack of applications should be configured. Along the way you can add any application or business details and procedures that are specific to your environment.

Each of these base types describe a solution to a typical recurring problem found when automating build and deployment processes. The processes that govern application life cycle are broken down into separate command workflows. Instances of base types are then combined together in order to coordinate the end to end life cycle process.

The rest of this document describes the Base Types that are used as the foundation for building your own automation pattern that will drive the build and deployment process for your specific application stack.

3. Base Types

This section provides a brief description to each of the base types. The intent of each base type is briefly stated along with its parent type and possibly related life cycle command. An explanation of the type inheritance hierarchy and composition model are also discussed later.

| Name | Description | Sub-type of | Life cycle command |
|----------|---|-------------|--------------------|
| Resource | The root of the class hierarchy establishing a common object addressing scheme. Also provides two properties, name and description. | | Not applicable |
| Assembly | An Assembly object is used to compose objects together into dependency hierarchies to represent part-whole relationships. Maintains a set of references to child objects that represent dependencies. | Resource | Not applicable |
| Setting | A Setting object describes a | Resource | Not applicable |

| | | | |
|----------------------------|--|----------------|-------------|
| | configuration property. | | |
| Managed-Entity | A Managed-Entity provides an interface to manage resource assemblies. Maintains a set of references to commands which can be executed. | Assembly | Install |
| Deployment | The Deployment type defines an object that maintains environment and dependency context, as well as, one that coordinates the installation and configuration processes. | Managed-Entity | Update |
| Package | The Package type defines an object that provides methods to create and install a package and maintain key properties that describe the package. | Managed-Entity | install |
| Node | The Node type represents a physical machine that hosts software deployments. Typically, a Node has one or more Deployment objects that represent that Node's software deployments. | Managed-Entity | dispatchCmd |
| Service | The Service type defines an object that provides methods to manage the runtime state and deployment of a software service. | Deployment | Update |
| Mediator | <i>TODO</i> | Deployment | dispatchCmd |
| Site | The Site type defines | Mediator | Update |

| | | | |
|-------------------------|---|------------|--------|
| | an object that provides methods to manage the runtime state and deployment of a set of related software services. | | |
| Updater | An Updater is responsible for interfacing with an application system (i.e., a site), configuring the package dependencies within the site and coordinating the installation, reconfiguration and restart of the deployment processes. | Mediator | Update |
| Builder | A builder is responsible for interfacing with the software build tool (e.g., Ant), configuring and executing the build, and then loading configured build artifacts into the repository. | Deployment | Build |

Table 1: Types

Inheritance Hierarchy

The base types are all derived from a common root type called Resource. Indeed, familiar base types, like Site and Updater are derived from other primitive base types like Deployment. Each base type inherits the properties and behavior (i.e., commands) from its parent. The diagram below describes the base types in the context of the type inheritance hierarchy.

Composition Hierarchy

Base types are composed to form automation patterns that form the basis of user models. The Assembly base type's *resources* property establishes the capability to define composition hierarchies used to model many kinds of whole-part relations.

The more specialized base types inherit the *resources* property to define dependencies relative to their role in the model. There are two kinds of recurring dependencies:

- concrete deployment dependency hierarchies: these describe the familiar physical objects - Nodes, Deployments and Packages - and how they are tied together to form the physical deployment model.
- logical whole-part hierarchies: these describe higher level objects - Updaters, Builders, Sites and Services - and how they are tied together to form the logical management model that provides simplified management.

Note: users can adjust these composition relationships by modifying the *resources* property constraint for their derived types.

4. Example

The Base Types were designed to allow you to quickly define build and deployment automation that follows the process outlined in the diagram below.

This process will move you from source code to a completely deployed and working application by utilizing two main types of automation, Builders - takes source code from your SCM system, builds it, packages it (perhaps in collaboration with a Package type), and registers the output with the deployment repository - and Updaters - coordinates the process to distribute packages from the repository, install them on local machines, generate context-specific configurations as needed, and execute the startup procedures.

The next diagram provides an example that illustrates a pattern to automate the build and deployment for a fictitious web site: **Headline News**. Each part of the end-to-end process is defined in terms of base types (shown in bold text).

In the diagram center, the Updater coordinates the two life cycle actions: **Build** and **Update**. The **Build** command is dispatched to the **Builder** which manages the build life cycle for the **Headline News** application components. The packages generated via the **Builder** are staged in the central repository. The **Update** command is dispatched to the **Site** which in turn relays the **Update** action to its **Services** deployed on their respective **Nodes**. The **Service** manages the deployment life cycle, which by default is defined to shutdown the service, download and install its package dependencies, configure the application components and then restart.

The figures representing the base types in the diagram are the actual icons used within Workbench. These icons aid identifying objects by their base type.

5. Documentation Format

Each base type (listed in the side bar) is documented following a standard format that organizes information into the following sections:

- Intent: Describes purpose and primary features of base type
- Problem: Describes problem base type solves
- Discussion: Describes typical use case of problem and proposed solution using the base type
- Structure: Describes model structure
- Example: Provides an example usage of the base type
- Check List: Lists steps to using base type
- Rules of Thumb: Additional caveats and suggestions when using base type