Tabula 3.0

Administrator's Guide

May 2009

Prepared by: BugBytes, Inc.,
Updated by CH2M HILL, Inc. (2005)
Updated by Brown and Caldwell (2008)

King County

Department of
Natural Resources and Parks
Wastewater Treatment Division

Table of Contents

Table of Contents 1
Introduction 5
Program Components 6
Program Deployment 8
  Windows Deployment 8
  Java Deployment 11
Configuration File Security 12
  Configuration File Signing Utility 12
  Deployment Options 12
Configuration File Formatting 13
  Cost Escalation Factors 13
  Domains 14
  Calculators 15
  Models 16
  Inheritance and Visibility 17
  General Layout 18
Bureau of Culvert Maintenance Example 19
  Initial Culvert Cost Calculation Requirements 19
  The Tabula Interface running the BCM's Model 20
  The XML Configuration file for the BCM Model 22
  Upgrading the BCM Model 25
  Modifying the BCM Model Configuration file 25
  Signing the updated BCM Model Configuration file 31
  Deploying the updated BCM Model Configuration file 32
  Results from the updated BCM configuration file 32
Configuration File Reference 34
  `<calculator>` Tag 35
  `<ceiling>` Tag 37
  `<cost>` Tag 39
  `<domain>` Tag 41

`<enrcci>` Tag 44
`<floor>` Tag 46
`<helptext>` Tag 48
`<icon>` Tag 50
`<ifgt>` Tag 51
`<ifle>` Tag 53
`<index>` Tag 55
`<inverse>` Tag 56
`<item>` Tag 58
`<lookup>` Tag 60
`<max>` Tag 62
`<min>` Tag 64
`<modelconst>` Tag 66
`<modelparam>` Tag 68
`<multiplier>` Tag 70
`<multiply>` Tag 72
`<number>` Tag 74
`<param>` Tag 76
`<power>` Tag 78
`<quantity>` Tag 80
`<range>` Tag 81
`<reference>` Tag 82
`<sum>` Tag 84
`<tabula>` Tag 86
`<ui>` Tag 87
`<unitcost>` Tag 88
`<var>` Tag 89
`<when>` Tag 91
`<xref>` Tag 93
`<year>` Tag 95
Data File Reference 96

`<data>` Tag 97
`<enrcci>` Tag 98
`<index>` Tag 100
`<item>` Tag 101
`<model>` Tag 102
`<param>` Tag 104
`<tabula>` Tag 106
Appendix A: Source Code Design and Layout 107
tabula.data.* Packages 107
tabula.ui Package 108
tabula.model.* Packages 109
tabula.security Package 109
tabula.util Package 109
Appendix B: Configuration File Modification Walkthrough 110
Step 1: Obtain a copy of the configuration file. 110
Step 2: Obtain a copy of the signing utility 110
Step 3: Verify JRE installation 110
Step 4: Modify the Configuration File 111
Step 5: Creating the *.sig file 111
Step 6: Update the Tabula Installation 112
Appendix C: Development Environment 113
Development Tools 113
Development Directory Layout 114
Appendix D: ENRCCI Values 116
Appendix E: Cost Calculation Atlas 117
Horizontally Drilled Pipe 117
Jack and Bore 131
Microtunnel 143
Microtunnel Template 160
Parallel Pipes 171
Pipe 182
Project 191

Pump Station 192
Reclaimed Water 197
Storage Facility 206
Trenched Conduit Template 212
Trenchless Conduit Template 237
Tunnel 250

Introduction

Tabula, developed for use by King County Staff and consultants, provides conveyance cost estimates at the planning level. It integrates information gathered through analysis of historical costs and other cost planning curves to provide budgetary planning estimates in a consistent and reproducible manner.

Over time, construction methods improve and other costs tend to change as a result of various causes. Eventually, the logic and formulas that Tabula uses to generate cost estimates become out of date and produce results that are too different from actual costs to be useful for budgetary planning purposes. For this reason, Tabula provides a relatively easy way to reconfigure the cost calculation logic and formulas.

This document describes the inner workings of the Tabula software and provides the information needed by an administrator to make updates to the cost calculation logic portion of the program. It starts with an overview of the basic program components, and follows with a detailed description of how the program is deployed on various target machine and operating systems. Next, it describes the methods by which the cost calculation logic is protected from tampering by the end user. It includes a hypothetical example. It finishes with the configuration and data file format references and some useful appendices.

Tabula leverages a multitude of computer technologies and design patterns; not all of these are explained in this document. It assumes that readers have a basic understanding of the eXtensible Markup Language (XML). This is required for editing the Tabula configuration file, which is written in XML. The bulk of the program itself is written in the Java programming language. Altering the source code requires at least an intermediate knowledge of the Java programming environment. The “bootstrap” program (program that loads the Tabula program in the Windows environment) requires knowledge of the Microsoft Windows Application Programming Interface (Win32-API) and how to launch the Java Virtual Machine from the operating system. Additionally the Tabula project utilizes the Wise Installation Environment for producing the Setup program; knowledge of this environment (or other Setup creation environments) is required to produce a setup program for a new distribution. Furthermore, the rationale behind the formulation of the cost calculations can be found in other references, including the *Conveyance System Cost Estimates Task 250 Report*.

Program Components

Tabula consists of multiple components. The primary executable program was written in the Java programming language, making it possible to deploy the program on a multitude of different operating systems and computers. A program written in Java is never actually compiled straight to native machine instruction code; instead, a Java program is compiled into what is called *bytecode*. Bytecode is a binary machine-independent representation of a software program that requires an additional piece of software to interpret the bytecode and execute the desired instructions. For the Java platform this interpreter is called the *Java Virtual Machine*. In order to execute Tabula on a target machine, a Java Runtime Environment (JRE) containing a Java Virtual Machine must be installed and available for use by Tabula. (Please see Sun Microsystem's website, http://java.sun.com, for more information on JRE's.)

The most significant deployment change between the previous versions of Tabula and the updated version is that the deployment module was developed in Visual BASIC and renamed from Tabula2 to Tabula3. This change allows all versions to be run side-by-side and makes installation much easier because Tabula3 is a new version rather than a replacement version. Other less important deployment changes are the use of a new "splash screen" to denote that the updated version is being accessed. Previous version utilized Setup Factory and Wise, for the creation of the setup package; Tabula3 utilized Install Creator to build the set up package. Because Tabula3 is most often installed on computers running 32-bit Microsoft Windows Operating Systems, the helper program, written in Visual BASIC, was used to automatically start up the Java Virtual Machine and instruct it to run the Tabula3 Program. This Tabula3 *launcher program* for the Windows Operating System makes Tabula3 appear to behave like a typical Windows application. The look and feel of the original launcher has been preserved to ensure that users who have previous experience with Tabula will immediately feel more comfortable using the updated version. The only change to this launcher was the replacement of the splash screen with an updated bitmap.

In addition to the Windows launcher, a Windows setup application has been written using the Install Creator system. Install Creator is a change from the previous installation systems. Install Creator is a change for the programming side of Tabula deployment but end users won't notice any difference. End users still run the setup program to install Tabula3 to their local desktop. This setup program automatically installs a private copy of the Java Virtual Machine along with the Tabula3 bytecode and makes the appropriate file associations and registry settings. The result is that installing and running Tabula3 on a computer running the Microsoft Windows Operating System requires no more effort than installing and running your average Windows application.

As mentioned above, Tabula provides an easy way to incorporate new information into the cost calculation logic. This is because the instructions detailing the cost calculations are separated from the compiled Java code and are encoded in an XML formatted text file. This XML configuration file contains instructions for the cost calculations for all of the Tabula Model components in a project. As a result of this separation of calculation and program user interface logic, changes in the cost calculation algorithms do not require any rewriting of Java code.

To prevent the arbitrary or accidental modification of the cost calculating logic, while loading the configuration file, Tabula also loads another file called the *signature* file. The signature file contains a digital fingerprint of the original configuration file. If the text in a configuration file on a target installation changes in any way, the fingerprint will no longer match and Tabula will refuse to run. The signature file is created with a utility (also written in Java) called the *SignTabula* Utility. This program creates the signature file that Tabula looks for when loading the configuration file. By using this scheme for protecting the integrity of Tabula's configuration, end users can look at, but not modify, the configuration file. It is important for caretakers of the Tabula program (and its configuration file) to avoid accidental distribution of this signing utility.

Additionally, Tabula relies upon some third party open-source Java libraries in order to efficiently read and write project and configuration files which are formatted in XML. It uses the Apache foundation's Xerces Parser version 1.1.3. It has not been tested with later versions of this library. For users that are running Tabula directly from Java, it is important to make sure that this library is in the Java *classpath*. This detail should not be a concern of the average Windows user, because the setup program installs the proper version of this library automatically.

Program Deployment

The deployment, or installation, of Tabula3 on the end user's target machine divides into two classes of installation: a Microsoft Windows installation or a Java Installation. As mentioned above, the Windows installation configures the system as necessary with the least possible interaction on the part of the user. However, as a result of this convenience, the resulting installation is a bit more complicated than using the program as a native Java application. This section describes these two types of deployments in detail.

Windows Deployment

Usually the Tabula3 Setup Program, named Tabula3Setup.exe, performs the installation of Tabula3 on a Windows platform. It creates the file directory structure expected by Tabula3; installs a private copy of a compatible Java Virtual Machine; copies over the needed Xerces Parser Java Library, Tabula3 class files (bytecode) and Tabula3 Windows Launcher program; creates *.tpr file associations with Tabula3; and installs the shortcuts to invoke the launcher program in the start menu. The setup program is a product of using the Setup Factory Installation System. Setup Factory is one of the many installation creation utilities used to create complicated Windows Setup Programs and it automatically handles the subtle differences between the Windows 9x and NT/2000 family of operating systems.

In the event that future maintainers of Tabula3 do not have Install Creator at their disposal or wish to switch to a different Windows Setup Program building platform (such as InstallShield) the following sections describe in detail the expected file layout and registry settings that Tabula3 expects to find when running in Windows. The main aspect of Tabula3 setup that needs to be considered is that previous versions of Tabula are not overwritten. Both versions can run side-by-side. This installation mechanism was chosen to ensure that important files are not accidentally deleted or overwritten.

Target File Directory Structure

The following image shows the basic directory layout of a Tabula installation on Windows:

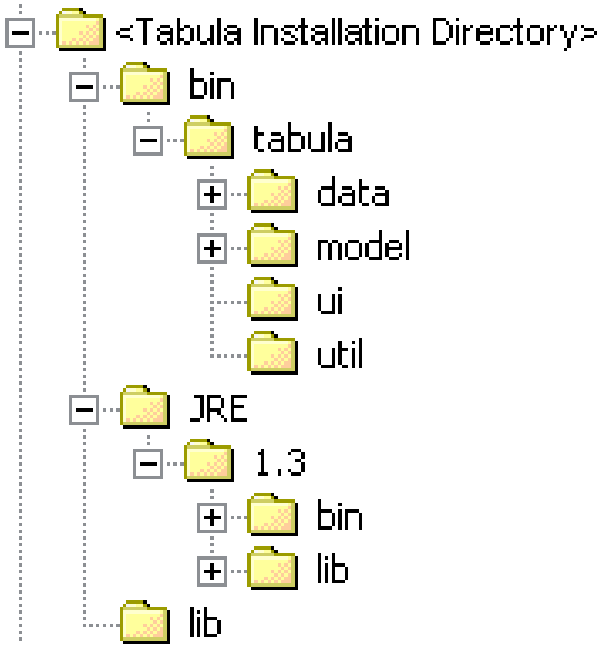


Figure 1 – Basic Windows Installation Directory Structure

The **<Tabula Installation Directory>** is the root directory for the Tabula Installation. The default path for this directory is `C:\Program Files\Tabula3` but the setup program allows the user to modify this value. The name and location of the root directory can vary, but its contents cannot. Aside from the child subdirectories, the root directory contains only one file, the Windows Launcher program named `Tabula3.exe`. This is the program that displays the initial splash screen, searches for a Java Virtual Machine, loads it, and then instructs it to run the Tabula bytecode.

The **bin** subdirectory and its subdirectories contain the **.class* files that store the bytecode that makes up the main portion of the Tabula program. For those savvy in Java, this is the root of the application classpath. The Windows Launcher searches for this directory as an immediate subdirectory of the directory that contains it. Moving this directory out of the root directory or changing the name of this directory will require changes to be made in the Windows Launcher Program (Tabula3.exe). The bin subdirectory contains one file at its root, `Tabula.class;` this is the entry point for the main portion of the Tabula program.

The **JRE** subdirectory is a private copy of a runtime version of the Java Virtual Machine Runtime Environment (JRE). Sun Microsystems permits this type of redistribution, for more information see the JRE documentation. No Windows registry settings are required for this. The directory structure and files contained in this subdirectory are specified by the JRE and cannot be changed. As with the bin subdirectory, the Windows Launcher program searches for the JRE subdirectory and then digs deeper for the desired dynamic linked library (DLL) that is uses to launch the Java Virtual Machine. If the JRE

subdirectory cannot be found, it will search the Windows registry, looking for a global installation of a JRE. If it finds it there, it will launch that one instead. Therefore it is possible for users to remove the JRE directory from the installation if they know that a proper JRE is already installed on the target machine.

The **lib** subdirectory stores Java Archive (JAR) Files containing additional third party libraries need by the Tabula program. Currently, Tabula requires only one, the Apache Group's Xerces XML Parser. It has the filename of `xerces.jar`. The Windows Launcher will add any JAR file found in this subdirectory to the Java Virtual Machine's classpath before executing the main Tabula program.

Tabula Startup Sequence on Windows

The previous sections briefly mentioned some of the interactions that the Tabula Windows Launcher program has with its environment; this section describes the actions taken by the program in a sequential step by step description. Roughly speaking the Tabula Windows Launcher performs the following tasks upon execution:

1. Displays the Tabula Splash Screen
2. Computes the Java *classpath*
3. Searches for a JRE installation
4. Loads the located jvm.dll library
5. Searches for third party libraries
6. Creates a new JVM with the appropriate classpath
7. Instructs the JVM to find the main Tabula class
8. Invokes the startup methods on the main Tabula class
9. Hides the Tabula Splash Screen
10. Passes control to the main Tabula program

The launcher program displays a splash screen as its first task. This is particularly important on slower machines because it can take some time for the JRE to load and initialize. In that way users can be certain that they did launch the program and will not attempt to launch it twice because "nothing happened."

Next, the launcher program computes the Java *classpath*. The classpath directs the JRE where the compiled bytecode of the Tabula program is located. The classpath includes the *<applicaton_path>\bin* directory as well as all of the Java Archive (JAR) Files stored in the *<application_path>\lib* directory.

The launcher program then searches for a copy of the JRE. It first looks for a private copy of the JRE stored in the bin subdirectory of the root application directory. Specifically, it searches for a file named *<application_path>\bin\hotspot\jvm.dll*. If the launcher finds this dynamic link library (DLL), it will assume that this is a valid JVM and will subsequently use it. If none is found, it will turn to the Windows registry. It looks for a global installation of the JRE in the Windows registry by searching for the value of the key *HKEY_LOCAL_MACHINE\Software\JavaSoft\Java Runtime*

Environment\CurrentVersion and then using this value (if found) to find the location of the installed JRE (please examine the source code for details.) If the search proved to be successful, the launcher program will continue, otherwise it will display an error message and terminate.

After locating a JRE, the program loads the JRE DLL into memory and makes the appropriate system calls to initialize the runtime with the proper classpath. If this is successful, it continues by instructing the JRE to load the main Tabula class file and call the startup methods on that object.

The launcher program then closes the splash screen window and transfers control to the JRE and main Tabula program. The launcher program (tabula.exe) still remains as the root process shown in the list of Windows processes, but the activity is taking place in the Java environment. Ending the tabula.exe process from the task manager will end the Tabula program.

Java Deployment

On platforms other than Windows, Tabula is deployed in a single Java Archive (JAR) File. This JAR file has a specific format recognized by Java and contains all the Java class files necessary to run the program (including library files). The name of the JAR file is *tabula.jar*.

Launching Tabula from a JAR file is relatively straightforward. Open a console session with the operating system, move to the directory containing the tabula.jar file, and issue the following command:

```
java -jar tabula.jar
```

Assuming a proper JRE (1.3 or later) is installed and available, the main Tabula window will appear in a few moments. If the JRE is version 1.2 (not tested, but may work) a slightly different command is required:

```
java -cp tabula.jar Tabula
```

Installation of the JRE is specific to the host operating system. Please see Sun Microsystem’s Java website, http://java.sun.com, for more information..

Configuration File Security

Tabula stores the instructions detailing the cost calculations in a file separate from the rest of the program. This makes it easy to update the cost calculations and eliminates the need to recompile the program when simple modifications become necessary. Additionally, for informational purposes, it is desirable to make these calculation instructions available in a read-only format for a clear understanding of the calculations involved.

To prevent unwanted modification of the configuration file, when Tabula loads the configuration file, it also looks for a *signature* file. The signature file contains a digital fingerprint or *digest* of the contents of the configuration file. (It uses the `java.security.MessageDigest` object to compute this digest using the "SHA" algorithm. Please see the source code for details.) If the configuration file changes in any way, the computed digest of the modified file will not match the one stored in the signature file provided in the installation process. If there is a mismatch, Tabula will refuse to load and will instruct the user to reinstall the software.

Configuration File Signing Utility

Generating a matching signature file for distribution is relatively straightforward. A utility called *SignTabula* can be run from the command line to generate a signature file. It is invoked by the following command:

```
java -jar SignTabula.jar <configuration_file>
```

Where `<configuration_file>` is the path to the configuration file to be signed. This Java program is packaged in the SignTabula.jar file and requires a proper JRE installation in order to run.

The signature file always has the same file name as the original file with ".sig" appended to it. Any time that the configuration file is modified it is important to run this utility to update the signature file. Therefore, any patch, update or redistribution of the configuration file will always need to include the corresponding signature file.

Deployment Options

At this time, no program has been written to deliver just a patch of the configuration file. If one were written for the Windows platform, it would first need to search for the Tabula install directory and then look for the files located at:

<application_directory>\bin\tabula\model\default_domains.xml
<application_directory>\bin\tabula\model\default_domains.xml.sig

These files are the configuration file and its signature respectively. Any automated patching program would need to replace these two files.

For deployment of Java installations, it is recommended to redistribute the tabula.jar file containing the updated configuration file. Creating an application to modify this JAR file would take much more effort than redistribution of the original file itself.

Configuration File Formatting

Now with the issues concerning the configuration file security out of the way, the discussion can move forward to the formatting of the configuration file itself. The data in the configuration file is expressed in XML. This format was chosen for multiple reasons: XML is easily readable by both machines and people, it is an industry standard, and third party open-source software exists for easily reading and writing such files. For more information on XML in general, please see the W3C organization's website (http://www.w3c.org).

The *dialect* of XML written in both the configuration and data files is specific to Tabula. Currently there is no Data Type Definition (DTD) or Schema of this dialect. The Tabula program itself enforces the grammatical rules of the XML file.

The data in the configuration file represents multiple abstract objects that Tabula uses to construct cost estimates. These top-level abstract objects include Cost Escalation Factors, Domains, Calculators, and Models. *Cost Escalation Factors* contain information such as the Engineering News Record Construction Cost Index (ENRCCI) values and inflation rate. *Domain* objects represent the physical properties of a unit object or model object as seen in the main Tabula window. This is the object that defines the parameters such as length, diameter, type, etc. It is also responsible for defining what is valid input for a property, such as minimum and maximum lengths. *Calculators* are the objects that contain the instructions for performing the cost calculations themselves. Domains and Calculators can also inherit properties and cost calculation formulas from previously defined Domains and Calculators.

Models are the abstract objects that reside typically only in data files. They contain actual parameter values for various instantiations of model types in a project. Models also include the ability to define custom sub domains and calculators for specialized models. These capabilities are available for users that wish to create customized objects that are not included in the default Tabula installation. This functionality was included in the original design of the system to allow for future expansion of the program but is not utilized at this time and is therefore not fully documented.

Cost Escalation Factors

The `<enrcci>` tag represents Cost Escalation Factors. An example of its usage follows:

```
<enrcci inflation="0.03" >
      <index year="1970" value="1408.2"/>
      <index year="1971" value="1569.78"/>
      <index year="1972" value="1679.62"/>
</enrcci>
```

The inflation rate (expressed as a decimal value, not a percentage) is specified in the `inflation` attribute of the `<enrcci>` tag. The `<enrcci>` tag may contain one or many `<index>` tags. Each `<index>` tag represents an ENRCCI value for the specified year. The `year` attribute designates the year and the `value` attribute specifies the ENRCCI value. The values do not need to be in any specific order. However, if multiple values are specified for the same year, the value that appeared last in the document order will be

used. This also applies to the `<enrcci>` tag itself. It can appear more than once in a configuration (or data) file, but it typically only needs to appear once.

Domains

The <domain> tag represents domains. An example of its usage follows:

```
<domain type="pipe" public="true"
                  template="trenched-conduit-template"
                  description="Pipe" >

      <param name="length" description="Length"
                        type="length"
                        units="ft"
                        default="100"/>
      <param name="type" description="Conduit Type"
                        type="enum"
                        default="gravity">
            <item display="Gravity Sewer" value="gravity"/>
            <item display="Force Main" value="force"/>
            <item display="High Head Force Main"
                                    value="high-force"/>
      </param>
      <param name="diameter"
                        description="Pipe Diameter"
                        type="enum" default="12">
            <item display="8 in."    value="8"  />
            <item display="10 in."   value="10"  />
            <item display="12 in."   value="12"  />
      </param>
</domain>
```

The example above shows a `<domain>` tag that defines a model of type “Pipe”. It inherits properties from the type `‘trenched-conduit-template’` (more on inheritance in the following sections) and defines three additional parameters, length, type and diameter. As can be seen in the example, parameters can be of different types (predefined in tabula) such as length, numbers and enumerations. A complete listing of the types appears in the Configuration File Reference. Parameters also have programmatic names (specified in the `name` attribute of the `<param>` tag), descriptive names (specified by the `description` attribute) and default values (specified by the `default` attribute.) For enumeration parameter types, additional sub-tags enumerating the possible values are also present. These tags specify both a display name and actual programmatic value.

Calculators

The <calculator> tag represents calculator objects. An example of its usage follows:

```
<calculator type="pipe" template="trenched-conduit-template"
            description="Pipe 1.0" enrcci-year="1999">
      <var name="outer-diameter" description="Outer Diameter"
                              units="ft" category="Geometry">
            <multiply>
                  <sum>
                        <when model-param="type"
                                    equals="gravity">
                              <lookup model-param="diameter"
                                    xref="concrete-odiam" />
                        </when>
                        ...
                  </sum>
                  <inverse><number value="12"/></inverse>
            </multiply>
      </var>
      <var name="excavation-depth" description="Excavation Depth"
                              units="ft" category="Geometry">
            <sum>
                  <modelparam name="cover-depth"/>
                  <reference name="outer-diameter"/>
                  <number value="1"/>
            </sum>
      </var>
      ....
      <cost name="dewatering-cost" description="Dewatering">
            <quantity units="lf">
                  <modelparam name="length"/>
            </quantity>
            <unitcost>
                  <sum>
                        <when model-param="dewatering"
                                    equals="minimal">
                              <lookup model-param="diameter"
                                          xref="trench-sump-
                                          dewatering-unitcost" />
                        </when>
                        <when model-param="dewatering"
                                    equals="significant">
                              <lookup model-param="diameter"
                                    xref="dewatering-well-
                                    unitcost" />
                        </when>
                  </sum>
            </unitcost>
      </cost>
</calculator>
```

Calculators are probably the most complex objects residing in the configuration file. They define calculated variables that can then be used, along with the original parameters, in cost calculations themselves. Both the variables (`<var>` tag) and the cost values (`<cost>` tag) can contain child tags that represent different types of mathematical

and logical operations. The tags that can appear as children of variable and cost tags are: `<ceiling>`, `<const>`, `<floor>`, `<ifgt>`, `<ifle>`, `<inverse>`, `<lookup>`, `<max>`, `<min>`, `<modelparam>`, `<multiply>`, `<number>`, `<power>`, `<reference>`, `<sum>` and `<when>`. For more information on these operators, please see the Configuration File Reference.

Calculators can also inherit from previously defined calculators. In the example above, this calculator inherits from the '`trenched-conduit-template'` calculator. Additionally, each calculator must be associated with a year which the calculations were based on. This is specified by the `enrcci-year` attribute. In this example the value is 1999.

Each type of cost is specified individually. Tabula will combine each cost item to compute the total cost for the model component.

Models

Model objects do not appear in the configuration file, but they are very prevalent in data files. The `<model>` tag represents model objects. An example of its usage follows:

```
<model type="pipe" name="Mainline">
        <data>
            <param name="construction-year" value="2006" />
            <param name="length" value="100" />
            <param name="type" value="gravity" />
            <param name="cover-depth" value="12" />
            <param name="backfill" value="imported" />
            <param name="disposal" value="none" />
            <param name="manhole-spacing" value="500" />
            <param name="existing-utilities" value="complex" />
            <param name="dewatering" value="minimal" />
            <param name="pavement-restoration" value="trench" />
            <param name="traffic" value="heavy" />
            <param name="row" value="none" />
            <param name="easment" value="none" />
            <param name="trench-safety" value="standard" />
            <param name="diameter" value="12" />
        </data>
 </model>
```

This model object is of the type “pipe” (specified by the `type` attribute) with the name “Mainline” (specified by the `name` attribute). The model tag has a sub-tag for instance data called the `<data>` tag. Inside the `<data>` tag are different `<param>` tags specifying the values of the parameter attributes of the model object as specified by its defining domain. If a `<param>` tag is missing for a particular attribute, Tabula will use the default specified in the `<domain>` tag. Specifying a value (`value` attribute of the `<param>` tag) that is outside the allowed domain of values for that particular attribute will cause an error when Tabula tries to load the file

Inheritance and Visibility

Shortly into the development of Tabula, it was noticed that many model objects had many commonalities and that it would be advantageous to build some sort of definition reuse scheme into the program. For example, the “pipe” and “parallel pipes” structures are both trenched technology. Many of the calculations regarding the cost of the trench are identical for both structures. With this in mind, it was decided to include the concepts of *inheritance* and *visibility* into Tabula’s configuration file definition format.

Inheritance and visibility work hand in hand to make the definition of cost calculations easier for the Tabula administrator. Any domain or calculator can first *inherit* the pattern described by any previously defined domain or calculator, and then be further refined by more parameter and cost definitions. Some of these ancestor objects, typically called *templates,* that are to be inherited from are not complete by themselves and should not be added to a Tabula project; setting the template object’s visibility to false specifies to the Tabula user interface that this is the case. For example, both the parallel pipe and standard pipe inherit properties from the trenched-conduit-template, but it makes no sense to have a trenched-conduit-template available for the end user to add to a project.

For example, consider first the definition:

```
<domain type="trenched-conduit-template"
            public="false"
            description="Trenched Conduit Template" >

      <param name="construction-year"
            description="Construction Year"
            type="year" default="2006"/>
      ...
```

Notice that the `public` attribute of the `<domain>` tag is set to `false`. This will prevent Tabula from displaying the “Trenched Conduit Template” item from insert menu.

Next examine the definition for pipes and parallel pipes:

```
<domain type="pipe" public="true"
            template="trenched-conduit-template"
            description="Pipe" >
      <param name="diameter" description="Pipe Diameter"
            type="enum" default="12">
      ...
```

and:

```
<domain type="ppipe" public="true"
            template="trenched-conduit-template"
```

```
        description="Parallel Pipes" >
<param name="diameter1" description="Pipe One Diameter"
        type="enum" default="12">
...
```

Both of these objects inherit some of their properties from the trenched conduit template. This inheritance relationship is specified in the `template` tag. Also, because Tabula should display these items in the insert menu, the `public` attribute is set to `"true"` as well.

Objects that inherit properties will also pass down these properties in the event a new child object inherits from this object. If an object contains a definition for a property already defined in an ancestor, it will overwrite the definition of that property. The inheritance scheme offered by Tabula allows for single inheritance, multiple inheritance is not supported.

General Layout

The root XML tag is `<tabula>`, both for configuration and data files. It is convention (and may be required in the future) that data files have a `model` attribute of the `<tabula>` tag set to `"true"`. Beyond the root tag, configuration files usually include `<enrcci>`, `<calculator>` and `<domain>` tags while the data files typically only contain `<enrcci>` and `<model>` tags. (Unless of course the modeler is using the advanced functionality of Tabula that allows for additional custom `<domain>` and `<calculator>` definitions.)

In general, if a calculated value (such as a variable, cost or year calculation) references another value such as a model parameter, constant, or calculated value, it must be defined before the reference. Also, circular references are not allowed. Templates must be defined before they are referenced as well.

Bureau of Culvert Maintenance Example

This section describes the lifecycle maintenance of a hypothetical Tabula deployment to the Bureau of Culvert Maintenance (BCM). This hypothetical department is solely responsible for all of the installation and maintenance of culverts within its jurisdiction. It uses Tabula as a tool to estimate project costs for new installations of culverts.

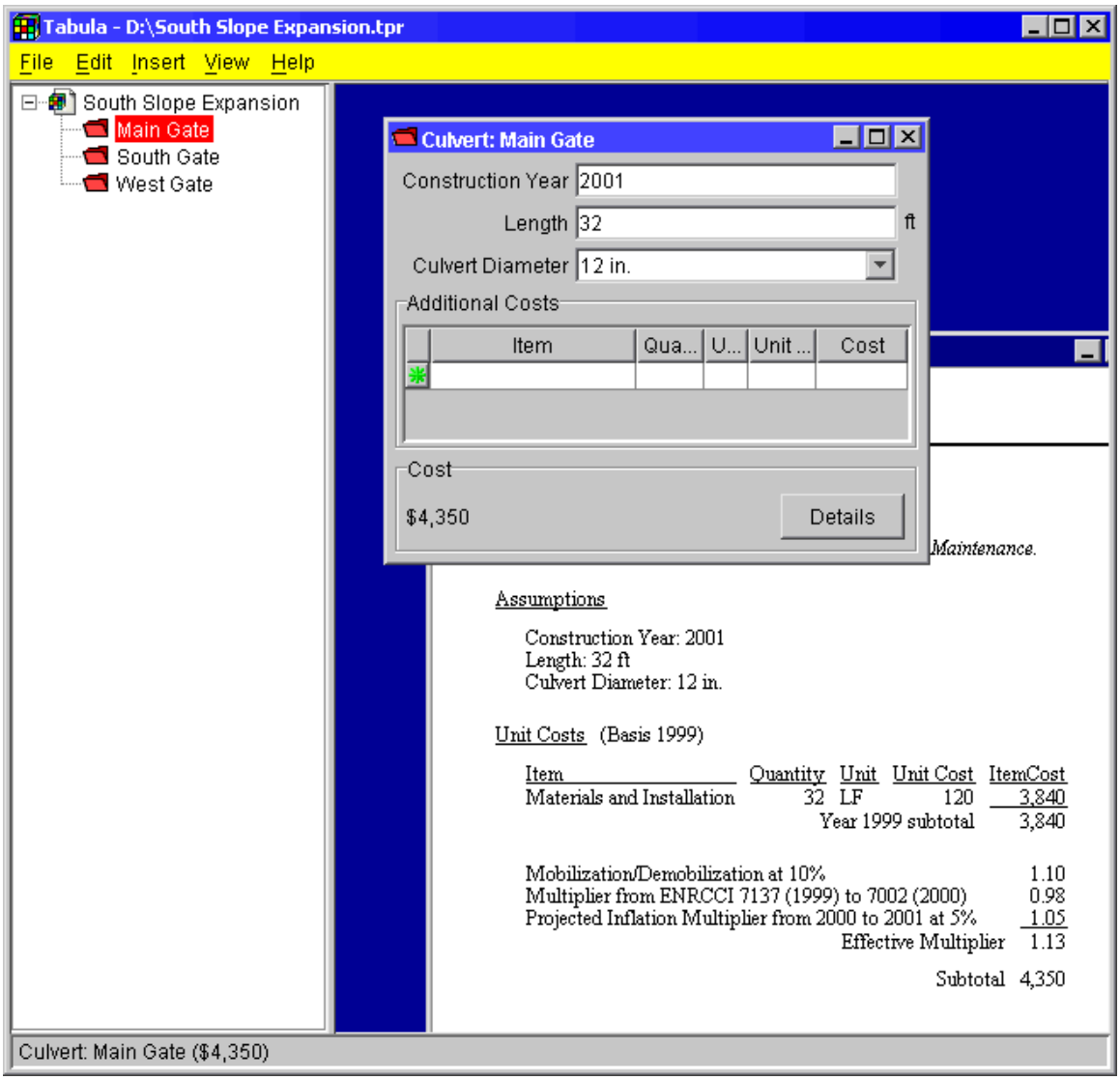
For demonstration purposes, this section utilizes the original configuration Tabula to show how the calculation requirements are translated into the XML code understood by Tabula. It then shows how the configuration is translated into a user interface and highlights the results of the cost calculations. Next, it introduces more complexity into the problem domain illustrating how the configuration can be updated to meet new requirements.

Initial Culvert Cost Calculation Requirements

At the time of the original Tabula deployment, the BCM required a very simple cost model. First, they only install and maintain culverts. Furthermore they have only used culverts of two sizes, 8-inch diameter culverts and 12-inch diameter culverts. Through years of experience they have determined that the cost of materials and installation averages out as $100 per linear foot for an 8-inch culvert and $120 per linear foot for a 12-inch diameter culvert (in year 1999 dollars). In the past, they have typically dispersed the displaced soil from installation on site or an employee has taken the soil home for their back yard. The BCM has never had to worry about spoils disposal. They typically add 10 percent to the cost of the culvert for mobilization of their equipment for planning purposes.

The Tabula Interface running the BCM's Model

One of the more recent proposed projects at the BCM is the South Slope Expansion. Recent development in the area around of the South Slope gated community has necessitated the installation of culverts across the driveways to the three access gates into the community. Three culverts will be installed, two 8-inch culverts at the minor gates and one 32-foot-long 12-inch culvert running under the driveway to the main gate. The following illustration displays a snapshot of the Tabula editing screen of the Project Engineers cost model for this project.



All Tabula cost models must have a root "Project"; in this case it is named "South Slope Expansion". It can be seen in the left pane of the main Tabula screen that the South Slope Project contains three sub-projects, each of them being a culvert. In the right portion of the screen, the editing window for the "Main Gate" culvert is displayed, with the corresponding detailed output report in the background.

The Main Gate culvert has a length of 32 feet, diameter of 12 inches and will be completed in the year 2001. The installation cost for the Main Gate culvert totals at $4,350. The detailed output report shows that the base material and installation cost runs at $3,840 (in year 1999 dollars). After a 10 percent markup for mobilization, a 2 percent reduction in the cost to convert to year 2000 dollars (using ENRCCI values) and a 5 percent markup to convert from year 2000 dollars to 2001 dollars using a 5 percent inflation rate yields the $4,350 total.

When saved to disk, the contents of the data file for the South Slope Project looks like:

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <tabula model="true" date="Tue Dec 11 14:07:41 EST 2006">
 3     <enrcci inflation="0.05" roundcalc="false"
 4             rounddisplay="true"/>
 5     <model type="project" name="South Slope Expansion">
 6         <model type="culvert" name="Main Gate">
 7             <data>
 8                 <param name="construction-year" value="2001"/>
 9                 <param name="length" value="32"/>
10                 <param name="diameter" value="12"/>
11             </data>
12         </model>
13         <model type="culvert" name="South Gate">
14             <data>
15                 <param name="construction-year" value="2001"/>
16                 <param name="length" value="14"/>
17                 <param name="diameter" value="8"/>
18             </data>
19         </model>
20         <model type="culvert" name="West Gate">
21             <data>
22                 <param name="construction-year" value="2001"/>
23                 <param name="length" value="14"/>
24                 <param name="diameter" value="8"/>
25             </data>
29         </model>
27         <data>
28             <param name="project-year" value="2001"/>
29             <param name="comments" value=""/>
30         </data>
31     </model>
32 </tabula>
33
```

The properties for the Main Gate culvert are visible in the file. The construction year value is mapped to a “parameter” for the Main Gate “model” of type “culvert” (line 7) that has the name “construction-year” (line 9). The “length” parameter has a value of 32 (line 10), and the “diameter” parameter has a value of 12 (line 11). The names of these parameters and how they are mapped to Tabula’s user interface are defined in the Tabula Configuration File.

The XML Configuration file for the BCM Model

Tabula's XML Configuration file provides the instructions for computing the cost of various Tabula model components (culverts in this example). It is an XML file and must follow the rules for XML files. Tabula defines an XML dialect or set of XML tags that it recognizes as instructions for computing cost calculations. On the Windows platform, the location for the configuration file and its signature file (more on that later) are:

<application_directory>\bin\tabula\model\default_domains.xml
<application_directory>\bin\tabula\model\default_domains.xml.sig

Where <application_directory> is the original installation directory of Tabula (typically "C:\Program Files\Tabula").

The contents of the configuration file for the BCM model discussed above are:

```
<tabula>
   <enrcci inflation="0.05">
      <index year="1996" value="6086.77"/>
      <index year="1997" value="6639.85"/>
      <index year="1998" value="6957.81"/>
      <index year="1999" value="7137.17"/>
      <index year="2000" value="7002.00"/>
   </enrcci>
   <domain type="project" description="Project" order="10">
      <ui class="tabula.model.project.ProjectInfoFrame" />
      <icon src="tabula/model/project/tree_icon.gif" />
      <param name="project-year"
             description="Project Year"
             type="year" default="2001"/>
      <param name="comments" description="Comments"
             type="comment" default=""/>
   </domain>
   <calculator type="project" description="Project"
               enrcci-year="1999"
               comment="These estimates are for
               the Bureau of Culvert Maintenance.">
      <year>
         <modelparam name="project-year"/>
      </year>
   </calculator>
   <domain type="culvert" description="Culvert"
           public="true" order="1">
      <param name="construction-year"
             description="Construction Year"
             type="year" default="2001"/>
      <param name="length" description="Length"
             type="length" units="ft" default="14"/>
      <param name="diameter" description="Culvert Diameter"
             type="enum" default="8">
         <item display="8 in."    value="8"  />
         <item display="12 in."   value="12"  />
      </param>
      <xref name="materials-and-install-unitcost"
             value-units="enum" result-units="$/LF" >
         <item value="8" result="100"/>
```

```
41          <item value="12" result="120"/>
42       </xref>
43    </domain>
44    <calculator type="culvert" description="Culvert 1.0"
45                enrcci-year="1999"
46                comment="These culvert estimates are for
47                the Bureau of Culvert Maintenance.">
48       <cost name="installation-cost"
49             description="Materials and Installation">
50          <quantity units="LF">
51             <modelparam name="length"/>
52          </quantity>
53          <unitcost>
54             <lookup model-param="diameter"
55                     xref="materials-and-install-unitcost" />
56          </unitcost>
57       </cost>
58       <multiplier name="mobilization-demobilization"
59                description="Mobilization/Demobilization at 10%">
60          <number value="1.1"/>
61       </multiplier>
62       <year>
63          <modelparam name="construction-year"/>
64       </year>
65    </calculator>
66 </tabula>
```

The following paragraphs describe the contents of this file in detail.

All XML files must contain a *root* tag, for Tabula configuration files this tag is always the `<tabula>` tag. The `<tabula>` tag typically contains `<enrcci>`, `<domain>` and `<calculator>` tags in the configuration file.

The `<enrcci>` tag (lines 2–8) defines the cost escalation parameters. Tabula provides two ways to describe escalating costs, the preferred option is to use ENRCCI values; these are defined using the `<index>` tags (lines 3–7). When ENRCCI values are not available, Tabula calculates cost escalation using a simple inflation rate. The default inflation rate (which the user can override in a project file) is specified by adding the “inflation” attribute to the `<enrcci>` tag (line 2). It can be seen above, that the default inflation rate is 5 percent for costs calculated outside the range for the years 1996 to 2000. For costs residing in this range of years, Tabula will use the ENRCCI values specified in the `<index>` tags (lines 3–7).

The BCM model is relatively simple in that it only has two types of entities to model, the Project and the Culvert. The Project is a required entity for all Tabula configurations. When creating a new project, Tabula will look for a model of type `project` and will create a new instance and use it as the root model object. Each model entity in Tabula requires two definitions: a *domain* and a *calculator*. Domains contain the definition of properties and constants associated with an entity. Calculators contain the definitions of the cost calculations associated with an entity.

The definition for the project entity spans lines 9 through 25. The `<domain>` tag provides the *domain* definition (lines 9–17) and the `<calculator>` tag provides the *calculator* definition.

The project type has two properties (also called parameters), the project year and a text description. The project year parameter is defined, like all parameters, using the `<param>` tag (lines 12–14). It is given the parameter type of `year` (for a list of parameter types please see the description of the `<param>` tag in the configuration file reference). The other parameter is a comment field (lines 15–16) and is given the `comment` type.

Additionally, the domain definition for the project type includes references to a display icon (line 11) and a user interface (line 10). If these values are specified, Tabula will attempt to load the customized editing screen and icon and display them instead of the default editing screen and icon. See the `<ui>` and `<icon>` descriptions in the reference for more information.

For the `project` type, the *calculator* does not do much, it simply acts as a container for other projects; therefore the `<calculator>` definition (lines 18–25) is relatively simple. It only contains the `<year>` tag (lines 22–24) that links the calculation of the construction year with the `'project-year'` parameter (defined in the domain on lines 12–14).

The more interesting definition is the one for the `culvert` type (lines 26–65). The domain definition (lines 26–43) contains three parameters and a cross-reference lookup table. It defines three parameters: `'construction-year'`, `'length'` and `'diameter'` (lines 28–37). The `construction-year` parameter is similar to the `project-year` property of the `project` type defined above. The `length` parameter (lines 31–32) has the type of `length` (specified by the `type`) attribute. The `diameter` parameter has a special type referred to as the `enum` type (short for enumerator). The `enum` type requires that the user chose among a set of discrete input values. In this instance, culvert sizes only come in 8-inch and 12-inch diameters. The child <item> tag specifies the possible selections for an enumerated `<param>` type (in this case lines 35–36). An item in an enumeration has both a display (specified by the `display` attribute) that is displayed by the user interface and a value (specified by the `value` attribute) that Tabula uses to perform calculations.

The domain definition for the `culvert` type contains one other interesting item, the `<xref>` tag. The `<xref>` tag defines a table of lookup constants. In this case the `<xref>` tag (lines 38–42) defines a table that maps culvert diameters with the material & installation cost for the associated size. The `<item>` child tag specifies the individual lookup entries in the table (lines 40–41).

The *calculator* definition provides more instructions for the `culvert` type than it did for the `project` type. It still has the year definition (lines 62–64) similar to the `project` type, but also includes a cost item for the material & installation cost plus a 10 percent markup multiplier.

The BCM describes the cost of a culvert primary being a function of its length and type. The `<cost>` tag (lines 48–57) encodes this cost calculation. Each cost item (also referred to as a line item cost) has two parts, a quantity and a unit cost. The `<quantity>` child tag specifies the calculations to compute the quantity (lines 50–52). The `<unitcost>` child tag specifies the calculations to compute the unit cost (lines 53–56). Also, each cost item specified using the `<cost>` tag appears on the output report under the heading specified by the `description` attribute, in this case `'Materials and Installation'`.

The quantity associated with the material & installation line item is simply a reference to the length property of the culvert (line 51, defined on lines 31–32).

The material & installation line item unit cost is a function of the culvert diameter. Because the unit costs associated with various culvert diameters were defined above using the `<xref>` tag (lines 38–42): it is a simple matter of looking up the proper material cost value using the `<lookup>` operator tag (lines 54–55).

Additionally, the BCM specified that a 10 percent markup should be applied to a culvert cost for mobilization of equipment and personnel, this markup appears as a multiplier on the cost subtotal. Multipliers are specified using the `<multiplier>` tag. The multiplier for the 10 percent markup would then be a constant value of 1.1 (lines 58–61).

Upgrading the BCM Model

The BCM has been using Tabula for many years and is mostly satisfied with the results. However, the planning department has begun to notice some discrepancies in the model when compared with actual costs. This coupled with some new environmental policies and the desire to install a new size of culvert (16-inch culvert) has prompted the BCM to conduct a review of their cost model.

After reviewing the BCM cost model, four changes were proposed:

1) ENRCCI values for the years 2001 and 2002 have been published (7120 and 7239 respectively); they should be incorporated into the cost calculations.
2) The average material & installation cost for 8-inch and 12-inch culverts has risen to $150 and $180 in year 2000 dollars respectively.
3) The bureau will start installing 16-inch culverts soon, it is estimated that material and labor to install this culvert will average $200 per linear foot (in year 2000) dollars.
4) New environmental regulations specify that all spoils must be transferred to the local landfill, it is expected that this will average at $30 per cubic yard (including transportation costs and disposal fees.) in year 2000 dollars.

The next section describes how these modifications are incorporated into the configuration file.

Modifying the BCM Model Configuration file

Typically the Tabula administrator keeps a backup copy of the configuration file on hand; but when that is not the case, a copy of the file can be obtained from any deployment of the program on a client machine. On Windows, the configuration file is located at:

`<application_directory>\bin\tabula\model\default_domains.xml`

where `<application_directory>` is the original installation directory of Tabula (typically "C:\Program Files\Tabula").

Once a copy of the BCM distribution of the configuration file is obtained, it can be edited using any standard text editor (such as notepad).

Updating the ENRCCI values

Adding new ENRCCI values is relatively simple. First locate the section of the document that contains the `<enrcci>` tag:

```
 2    <enrcci inflation="0.05">
 3       <index year="1996" value="6086.77"/>
 4       <index year="1997" value="6639.85"/>
 5       <index year="1998" value="6957.81"/>
 6       <index year="1999" value="7137.17"/>
 7       <index year="2000" value="7002.00"/>
 8    </enrcci>
```

For each new ENRCCI value, insert a new `<index>` tag. The `year` attribute corresponds to the year and the `value` attribute corresponds to the new index value:

```
  7       <index year="2000" value="7002.00"/>
7.1       <index year="2001" value="7120.00"/>
7.2       <index year="2002" value="7239.00"/>
  8    </enrcci>
```

That completes the update to the ENRCCI values.

Updating unit costs for Culverts

Updating the unit costs of culvert materials & installation is straightforward with one potential complication. The new cost values are in terms of year 2000 dollars. Therefore we must also specify that the costs computed by the associated calculator are in year 2000 dollars or re-scale the cost values to match the cost basis year for the calculator (which would be year 1999 dollars.) Because the calculator is simple it is easier to modify the *basis year* for the calculator. This will be done first. Locate beginning lines of the calculator definition for a culvert:

```
 44    <calculator type="culvert" description="Culvert 1.0"
 45                enrcci-year="1999"
 46                comment="These culvert estimates are for
 47                the Bureau of Culvert Maintenance.">
 48       <cost name="installation-cost"
 49             description="Materials and Installation">
```

Notice the `enrcci-year` attribute on line 45, it specifies that this calculator will produce results in year 1999 dollars. Change this to 2000.

```
 44    <calculator type="culvert" description="Culvert 1.0"
 45                enrcci-year="2000"
 46                comment="These culvert estimates are for
 47                the Bureau of Culvert Maintenance.">
 48       <cost name="installation-cost"
 49             description="Materials and Installation">
```

Next the unit costs for the 8-inch and the 12-inch culverts must be updated. Remember that these values were stored in a cross-reference table; locate the cross-reference table for material & installation costs for the `culvert` type:

```
38      <xref name="materials-and-install-unitcost"
39             value-units="enum" result-units="$/LF" >
40         <item value="8" result="100"/>
41         <item value="12" result="120"/>
42      </xref>
```

The modification is simple, just update the associated `result` attributes:

```
38      <xref name="materials-and-install-unitcost"
39             value-units="enum" result-units="$/LF" >
40         <item value="8" result="150"/>
41         <item value="12" result="180"/>
42      </xref>
```

Adding the 16-inch Culvert to the Model

Two places are affected by adding a new culvert size to the model. A new enumerated value must be added to the `diameter` property of the `culvert`, and the associated unit cost for the new culvert must be added to the materials & installation cross-reference table.

First, locate the definition of the enumeration of valid culvert diameters:

```
33      <param name="diameter" description="Culvert Diameter"
34             type="enum" default="8">
35         <item display="8 in."    value="8"  />
36         <item display="12 in."   value="12"  />
37      </param>
```

To add a new culvert diameter, simply insert a new `<item>` tag. The `value` attribute should be something unique (preferably `'16'`). The `display` attribute should be something that is appropriate for listing by the Tabula editing screen:

```
 33      <param name="diameter" description="Culvert Diameter"
 34             type="enum" default="8">
 35         <item display="8 in."    value="8"  />
 36         <item display="12 in."   value="12"  />
36.1        <item display="16 in."   value="16"  />
 37      </param>
```

The next step is to add the 16-inch material & installation cost to the lookup table:

```
 38      <xref name="materials-and-install-unitcost"
 39             value-units="enum" result-units="$/LF" >
 40         <item value="8" result="150"/>
 41         <item value="12" result="180"/>
41.1        <item value="16" result="200"/>
 42      </xref>
```

That completes the steps to add a new culvert size to the BCM model.

Adding Spoils Disposal Cost

Adding the spoils disposal cost is probably the most involved modification to the configuration file that is required. First, the volume of the spoils must be calculated (it would also be nice to show that intermediate computation in the detailed output report) and then a cost associated with that volume must be computed.

The amount of spoils from installing the culvert is roughly the volume of the space the culvert occupies (0.25 x Pi x [culvert diameter]^2 x [culvert length]). Because it is desirable to display this value in the details section of the output report, this numerical computation will first be stored in a *variable*. The following should be inserted inside the <calculator> tag for the `culvert` type:

```
 47.1        <var name="spoils-volume" description="Spoils Volume"
 47.2                 units="CY" category="Geometry">
 47.3           <multiply>
 47.4              <number value="0.25"/>
 47.5              <number value="3.14"/>
 47.6              <power exponent="2">
 47.7                  <multiply>
 47.8                     <modelparam name="diameter"/>
 47.9                     <inverse><number value="12"/></inverse>
47.10                  </multiply>
47.11              </power>
47.12              <modelparam name="length"/>
47.13              <inverse><number value="27"/></inverse>
47.14           </multiply>
47.15        </var>
```

This *variable* will appear on the detailed output with the label “Spoils Volume” (specified by the `description` attribute, line 47.1) under the category of “Geometry” (specified by the `category` attribute, line 47.2). The `<multiply>` tag multiplies the factors of the equation together (lines 47.3–47.14). The `<number>` tag is used for numerical constants. Additionally it should be noted that the `diameter` parameter is in units of inches, it must be converted to feet, and then the whole result is converted from cubic feet to cubic yards.

The next step is to make a decision about where to store the unit cost value for the spoils disposal. There are two options: the first is that it would be directly referenced in the *calculator* through use of the `<number>` tag; the other option is to specify it as a constant in the associated *domain* and then reference that value using the `<modelconst>` tag. This example will use the second option. The unit cost for spoils disposal is $30 per cubic yard. This is inserted into the `culvert` domain as:

```
42.1        <const name="spoils-disposal-unitcost"
42.2                 type="cost" value="30" units="$/CY" />
```

Finally, the instructions for computing the spoils disposal line item can be inserted into the file:

```
57.1        <cost name="spoils-disposal-cost"
57.2              description="Spoils Disposal">
57.3           <quantity units="CY">
57.4              <reference name="spoils-volume"/>
57.5           </quantity>
57.6           <unitcost>
57.7              <modelconst name="spoils-disposal-unitcost"/>
57.8           </unitcost>
57.9        </cost>
```

The line item cost references the previously defined spoils disposal unit cost (line 57.8) for the unit cost of the spoils disposal. The volume of spoils was already computed and is referenced using the `<reference>` tag (line 57.4).

This completes the modifications to the configuration file that will achieve the desired changes to the BMC model, the contents of file in its entirety are now:

```
  1 <tabula>
  2    <enrcci inflation="0.05">
  3       <index year="1996" value="6086.77"/>
  4       <index year="1997" value="6639.85"/>
  5       <index year="1998" value="6957.81"/>
  6       <index year="1999" value="7137.17"/>
  7       <index year="2000" value="7002.00"/>
7.1       <index year="2001" value="7120.00"/>
7.2       <index year="2002" value="7239.00"/>
  8    </enrcci>
  9    <domain type="project" description="Project" order="10">
 10       <ui class="tabula.model.project.ProjectInfoFrame" />
 11       <icon src="tabula/model/project/tree_icon.gif" />
 12       <param name="project-year"
 13              description="Project Year"
 14              type="year" default="2001"/>
 15       <param name="comments" description="Comments"
 16              type="comment" default=""/>
 17    </domain>
 18    <calculator type="project" description="Project"
 19                enrcci-year="1999"
 20                comment="These estimates are for
 21                the Bureau of Culvert Maintenance.">
 22       <year>
 23          <modelparam name="project-year"/>
 24       </year>
 25    </calculator>
 26    <domain type="culvert" description="Culvert"
 27            public="true" order="1">
 28       <param name="construction-year"
 29              description="Construction Year"
 30              type="year" default="2001"/>
 31       <param name="length" description="Length"
 32              type="length" units="ft" default="14"/>
 33       <param name="diameter" description="Culvert Diameter"
 34              type="enum" default="8">
 35          <item display="8 in."    value="8"  />
```

```
   36          <item display="12 in."   value="12"  />
 36.1          <item display="16 in."   value="16"  />
   37       </param>
   38       <xref name="materials-and-install-unitcost"
   39              value-units="enum" result-units="$/LF" >
   40          <item value="8" result="150"/>
   41          <item value="12" result="180"/>
 41.1          <item value="16" result="200"/>
   42       </xref>
 42.1       <const name="spoils-disposal-unitcost"
 42.2                type="cost" value="30" units="$/CY" />
   43    </domain>
   44    <calculator type="culvert" description="Culvert 1.0"
   45                enrcci-year="2000"
   46                comment="These culvert estimates are for
   47                the Bureau of Culvert Maintenance.">
 47.1       <var name="spoils-volume" description="Spoils Volume"
 47.2                units="CY" category="Geometry">
 47.3          <multiply>
 47.4             <number value="0.25"/>
 47.5             <number value="3.14"/>
 47.6             <power exponent="2">
 47.7                 <multiply>
 47.8                    <modelparam name="diameter"/>
 47.9                    <inverse><number value="12"/></inverse>
47.10                 </multiply>
47.11             </power>
47.12             <modelparam name="length"/>
47.13             <inverse><number value="27"/></inverse>
47.14          </multiply>
47.15       </var>
   48       <cost name="installation-cost"
   49             description="Materials and Installation">
   50          <quantity units="LF">
   51             <modelparam name="length"/>
   52          </quantity>
   53          <unitcost>
   54             <lookup model-param="diameter"
   55                     xref="materials-and-install-unitcost" />
   56          </unitcost>
   57       </cost>
 57.1       <cost name="spoils-disposal-cost"
 57.2             description="Spoils Disposal">
 57.3          <quantity units="CY">
 57.4             <reference name="spoils-volume"/>
 57.5          </quantity>
 57.6          <unitcost>
 57.7             <modelconst name="spoils-disposal-unitcost"/>
 57.8          </unitcost>
 57.9       </cost>
   58       <multiplier name="mobilization-demobilization"
   59                description="Mobilization/Demobilization at 10%">
   60          <number value="1.1"/>
   61       </multiplier>
   62       <year>
   63          <modelparam name="construction-year"/>
   64       </year>
```

```
   </calculator>
</tabula>
```

The next step is to deploy the updated configuration file.

Signing the updated BCM Model Configuration file

Before Tabula will read a new configuration file, a *signature* file that matches the configuration file must be created. This is done to prevent end users from casually modifying the Tabula configuration. If Tabula finds that the *signature* file does not match the currently installed configuration file, it will refuse to load.

SignTabula is the utility that creates the signature file needed by Tabula. It is a Java program and is distributed in a file named `SignTabula.jar`. The most current version of the signing utility can be found at:

```
<root>\KingCo-CSI\Projects\tabula\SignTabula.jar
```

where `<root>` is the development project root directory or drive letter of the CDROM containing the Tabula development CD. It requires the Java Runtime Environment (JRE) version 1.2 or greater to be installed on the computer. For more information about the signing utility, please see Appendix B.

To generate the signature file, place the configuration file and the `SignTabula.jar` file in the same directory. Open a command window (DOS prompt), move to the same directory and type (for JRE version 1.3 and above):

`java -jar SignTabula.jar <configuration_file>`

where `<configuration_file>` is the path to the configuration file to be signed (typically `default_domains.xml`).

If the JRE version is 1.2.x, then the command is slightly modified to the following:

`java -cp SignTabula.jar tabula.security.SignTabula <configuration_file>`

If all goes well, the program should respond with:

```
Reading ... Digesting ... Writing ...Done.
```

Now there should be a file with the same name as the configuration file appended with "`.sig`" in the same directory as the updated configuration file. For example if the file was named `default_domains.xml` (as it was in this example), there should be an additional file named `default_domains.xml.sig`.

Deploying the updated BCM Model Configuration file

There are two options in regards to deploying an updated configuration file: copying the file and its signature into the Tabula application directory on a target computer; or creating a new setup program that contains the updated files. Although making a new installation program is the preferred option, it is rather involved and will not be discussed here.

The quickest way to reconfigure an installation of Tabula is to overwrite the current configuration files with the new ones. Copy the updated `default_domains.xml` and `default_domains.xml.sig` files to:

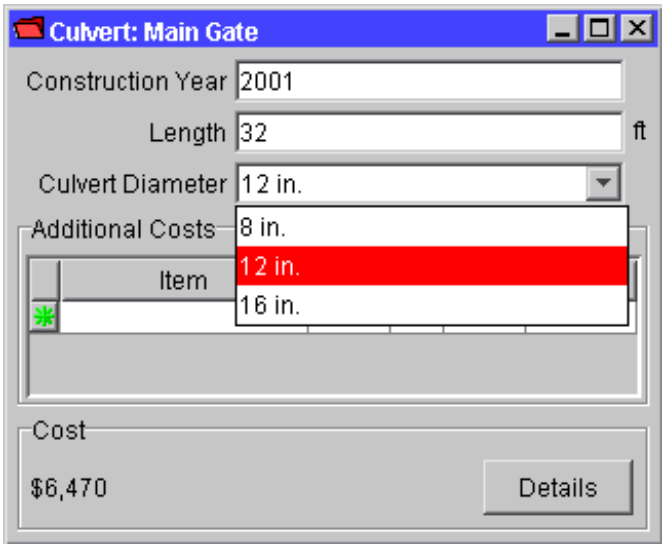
```
<application_directory>\bin\tabula\model\default_domains.xml
<application_directory>\bin\tabula\model\default_domains.xml.sig
```

where <application_directory> is the original installation directory of Tabula (typically "C:\Program Files\Tabula").

After copying the files, restart Tabula and the changes will take affect.

Results from the updated BCM configuration file

Now that the configuration has been updated, the user interface will show the additional culvert diameter in the dropdown list on the editing screen:



And the detailed output report for the Main Gate culvert has changed as well:

Main Gate Cost Detail

Cost Calculations for Culvert: **Main Gate**

Project year: 2001

These culvert estimates are forthe Bureau of Culvert Maintenance.

Assumptions

Construction Year: 2001
Length: 32 ft
Culvert Diameter: 12 in.

Geometry

Spoils Volume 0.93 CY

Unit Costs (Basis 2000)

Item	Quantity	Unit	Unit Cost	ItemCost
Materials and Installation	32	LF	180	5,760
Spoils Disposal	1	CY	30	28
			Year 2000 subtotal	5,790

Mobilization/Demobilization at 10%	1.10
Multiplier from ENRCCI 7002 (2000) to 7120 (2001)	1.02
Effective Multiplier	1.12
Subtotal	6,470

Total: $6,470

The report shows the amount of spoils that will be generated by the installation of the culvert under the category *Geometry*. The new *Spoils Disposal* line item is present. Also, the cost escalation multiplier now uses ENRCCI values to convert from the new Cost basis of 2000 to the cost year 2001.

This concludes the Bureau of Culvert Maintenance example. For more information on the details of the components of the configuration file, please see the following Configuration File Reference.

Configuration File Reference

Each abstract object appearing in the configuration file has a corresponding XML tag that appears in the configuration file. There are specific rules about where these tags can appear, in what order, and in what context. This section is the reference describing the format of these tags in detail, what children tags they have, and what attributes must be specified.

`<calculator>` *Tag*

The `<calculator>` tag and its enclosed tags define the calculations used to compute the estimated cost of an individual model object. The calculator definition can inherit instructions from a previously defined calculator and act as a *template* for calculators yet to be defined.

A calculator has three major sections: variables, costs and the year specifier. Variables are intermediate calculations that are necessary to simplify some of the more complex cost formulas. They can also be displayed on the detailed output screen for informational purposes. Costs specify the actual cost calculations for each cost item (a cost item has both a unit cost and number of units.). The year specifier indicates how the project year is determined from the model properties.

Both the `<tabula>` tag and the `<model>` tag can enclose the `<calculator>` tag. Typically, multiple `<calculator>` tags appear only as children of the root `<tabula>` tag. Usage of the `<calculator>` tag inside a `<model>` tag is advanced functionality and is not discussed in this document.

Parent Tags

Parent Tag	Comment
`tabula`	This is the typical parent for the `<calculator>` tag. Most calculators have “global” scope.

Child Tags

Child Tag	Comment
`var`	Specifies variable calculations that other calculations can be based upon, such as trench width as a function of pipe diameter. The trench width can then be used to calculate the spoils.
`cost`	Encodes an actual cost calculation. Each cost calculation has two parts: The first part is the quantity of the item in question, such as 200 cubic feet of spoil. The second part is the unit cost of the item such as $20.00 per cubic feet of hauled spoil.
`year`	Specifies how the model objects' completion year is calculated. This typically is a direct reference to a model parameter such as "Project Year."
`multiplier`	Specifies an additional multiplier to be applied to the cost subtotal of unit costs. This is in addition to the multiplier applied for conversion of project costs to different year basis. One such example is adding 10 percent for mobilization and demobilization of equipment.

Attributes

Attribute	Required	Purpose
`type`	yes	Specifies the type name of the model (defined by the `<domain>` tag) that this calculator applies to.
`description`	yes	Specifies a brief display name for the calculator.
`enrcci-year`	yes	Specifies the ENRCCI basis year that the following calculations are based on. This value is combined with the project completion year to scale the computed cost of the model object.
`template`	no	Specifies a calculator to inherit from. Any definitions specified in the body of this calculator with the same identifiers of the template calculator replace the original definition found in the template.
`comment`	no	If specified, this text appears at the top of the detailed cost calculation report for all items of this type.

`<ceiling>` *Tag*

The *ceiling* operation computes the smallest integer value greater than or equal to the value computed that is the *sum* of the results of immediately contained child operators.

The `<ceiling>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example computes the number of manholes required by a pipe in order to determine the cost of installing manholes.

```
<cost name="manhole-cost" description="Manholes">
      <quantity units="MH">
            <ifgt reference="manhole-spacing" value="0">
                  <ceiling>
                        <multiply>
                              <modelparam name="length"/>
                              <inverse>
                                    <modelparam
                                    name="manhole-spacing"/>
                              </inverse>
                        </multiply>
                  </ceiling>
            </ifgt>
      </quantity>
      ...
</cost>
```

In this example, the number of manholes is computed by taking the length of the pipe, dividing it by the manhole spacing and then rounding up to the next whole number by using the `<ceiling>` *operator*.

`<cost>` *Tag*

The `<cost>` tag is enclosed inside a `<calculator>` tag. It defines the calculations that compute an individual line item cost for a model object and is one of the three types of children tags of the `<calculator>` tag. A cost item has both a unit cost and number of units required, represented by the `<quantity>` and `<unitcost>` child tags respectively.

A definition of a unit cost by using the `<cost>` tag overrides any inherited cost calculation defined in any parent templates used by the model object calculator description.

Parent Tags

Parent Tag	Comment
`calculator`	This is the only parent for the `<cost>` tag.

Child Tags

Child Tag	Comment
`quantity`	Contains child operator tags defining the calculations for computing the quantity value for this cost item.
`unitcost`	Contains child operators that define the calculations for computing the unit cost value for this cost item.

Attributes

Attribute	Required	Purpose
`name`	yes	A unique programmatic name of the cost item. If this name matches the name of a previously defined cost item in a parent template, this definition will override the previous definition. It is an error to specify `<cost>` items with the same name inside the same calculator.
`description`	yes	The description of the cost item that will be displayed on detailed output reports.

Example

The following illustrates the cost calculation for launch shaft excavation.

```
<cost name="ls-excavation-cost" description="Launch Shaft Excavation">
      <quantity units="CY">
            <reference name="ls-volume"/>
      </quantity>
      <unitcost>
            <modelconst name="shaft-excavation-unitcost"/>
      </unitcost>
</cost>
```

The launch shaft excavation cost contains a quantity of material to be excavated, measured in cubic yards, with a unitcost that was specified by a previously defined constant model parameter.

`<domain>` *Tag*

The `<domain>` tag describes the properties and constants associated with a model object. It provides the definition for input parameters, their types and limitations. It is also can contain information about constants, such as the unit cost of hauling spoil and lookup or cross-reference tables for enumerated items, such as the cost per linear foot of concrete pipe as a function of diameter.

Domains can inherit definitions from previously defined domains called *templates*. A template must be previously defined in the configuration file before it can be referenced from another domain.

Additionally, domains can be specified as public or private. Public domains can be inserted into projects and are visible from the Insert menu in Tabula. Private domains are typically solely used as *templates* for other more complex domain definitions. By specifying a domain as private, common properties among a class of model objects can be defined in one place without having to resort to duplicating the information in multiple places throughout the configuration file.

Parent Tags

Parent Tag	Comment
`tabula`	This is the typical parent for the `<domain>` tag. Most domains have “global” scope.
`model`	When the `<domain>` tag appears as a child of a model object, the scope and visibility of the parameters for the type specified by this domain are only visible to child models of the enclosing model. In this way domains can be specialized in different contexts. This only works for user-defined domains and is considered advanced functionality that is not discussed further in this document. The `<model>` tag does not appear in the configuration file. It appears in the data file.

Child Tags

Child Tag	Comment
`const`	Tag containing definitions of constant values such as fill unitcost or paving unit cost. The associated calculators can reference these values by using the `<modelconst>` *operator* tag.
`param`	Defines variable parameters that must be defined by user input for this model. These values of these parameters for a given model instance are specified by using the `<param>` tag inside a `<data>` tag inside the associated `<model>` tag in a data file.
`xref`	Associates pairs of values in a lookup table such as linear cost of concrete pipe as a function of its diameter. The associated calculators can then use this lookup table through the `<lookup>` *operator* tag.
`ui`	Specifies a custom Java Class Object that extends `javax.swing.JinternalFrame` and provides a public constructor that takes a `tabula.ui.ModelRelay` and `tabula.ui.ViewManager` as arguments. This is the way custom editing screens for new models can be added to Tabula. Implementation of this feature requires a working knowledge of the Java source code for Tabula.
`icon`	Specifies the path to a custom Icon to represent this associated model in the Tabula user interface. This is typically a 16x16 color image stored in a file using the GIF format and located conveniently in the Java classpath. It is loaded using the Java `ClassLoader.getSystemResource` method. The icon appears next to the model name in the insert menu as well as associated with the default editing screen (unless a customized editing screen is specified using the `<ui>` tag).

Attributes

Attribute	Required	Purpose
`type`	yes	Specifies the programmatic name of a model type. The name is global. Defining a domain with the name of an existing domain (which can only be done in system configuration files) obscures the original definition of the model domain.
`public`	no	Specifies if this domain publicly creatable. Public domains appear on Tabula's insert menu. Valid values are `"true"` and `"false"`. If not specified, `"true"` is assumed.
`template`	no	Specifies a model domain definition to inherit from. Any definitions specified in the body of this domain with the same identifiers of the template domain replace the original definition found in the template.
`description`	yes	Description of this model type that will be displayed on the insert menu and output reports.
`order`	no	Specifies a numerical priority when sorting model types for display on the insert menu. Lower numbers have higher priority over higher numbers. If not specified, placement order in the insert menu is undefined.

Example

The following is an example the simple domain definition for the project type.

```
<domain type="project" description="Project" order="10">
      <ui class="tabula.model.project.ProjectInfoFrame" />
      <icon src="tabula/model/project/tree_icon.gif" />
      <param name="project-year"
            description="Project Year"
            type="year" default="2001"/>
      <param name="comments"
            description="Comments"
            type="comment" default=""/>
</domain>
```

The project is a simple model type that is primarily used for grouping. It only has two input parameters, the project year (specified to be the year type) and the comments (a free text entry). The definition also indicates to Tabula that it should use a particular icon and user interface for editing this object. If the `<ui>` or `<icon>` tags are not specified, Tabula's default editing screen and icon would be supplied to the user.

`<enrcci>` *Tag*

The `<enrcci>` tag represents Cost Escalation Factors. It can specify both Engineering News Record Construction Cost Index (ENRCCI) values as well as the inflation rate where ENRCCI values are not known. The tag is typically the first child of the `<tabula>` tag (in both configuration and data files), but can appear anywhere as a child of the `<tabula>` tag. It will be ignored if appearing as a child of any other tag. If multiple values for a given year are specified (either in the same `<enrcci>` tag or spanning across multiple tags) the latest specification in document order will be the one used by Tabula.

Parent Tags

Parent Tag	Comment
`tabula`	This is the only valid parent for the `<enrcci>` tag.

Child Tags

Child Tag	Comment
`index`	Specifies a particular ENRCCI value entry for a given year.

Attributes

Attribute	Required	Purpose
`inflation`	yes	The numerical value for the inflation rate to be applied to calculations where ENRCCI values are not known. This value is specified decimal value, not percentage (unlike the Tabula UI itself, which requires the value to be specified as a percentage).
`roundcalc`	no	Specifies that number values should be rounded to three significant figures between calculations. Specify the value `"true"` for turning this feature on and `"false"` for turning it off. The default is `"false"`. If specified, this value overrides default user settings.
`rounddisplay`	no	Specifies that numerical results should be rounded to three significant figures for display purposes. Specify the value `"true"` for turning this feature on and `"false"` for turning it off. The default is `"false"`. If specified, this value overrides default user settings.

Example

An example of the `<enrcci>` tag follows:

```
<enrcci inflation="0.03" >
      <index year="1970" value="1408.2"/>
      <index year="1971" value="1569.78"/>
      <index year="1972" value="1679.62"/>
</enrcci>
```

The inflation rate (expressed as a decimal value, not a percentage) is specified in the `inflation` attribute of the `<enrcci>` tag. The `<enrcci>` tag may contain one or many `<index>` tags. Each `<index>` tag represents an ENRCCI value for the specified year. The `year` attribute designates the year and the `value` attribute specifies the ENRCCI value. The values do not need to be in any specific order. However, if multiple values are specified for the same year, the value that appeared last in the document order will be used.

`<floor>` *Tag*

The `<floor>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *floor* operation computes the largest integer value less than than or equal to the value computed that is the *sum* of the results of immediately contained child operators.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example evaluates to the value of 3.

```
<floor>
      <multiply>
            <number value="7"/>
            <inverse>
                  <number value="3"/>
            </inverse>
      </multiply>
      <number value="1"/>
</floor>
```

`<helptext>` *Tag*

The `<helptext>` tag is an optional tag that specifies context sensitive help text to be displayed by Tabula when the user focus is on a specified field in a model editing screen. This tag appears as a child of the `<param>` tag in configuration files. It contains simple text nodes and HTML tags for its children. The specified HTML should not be too complex; more complex HTML will put a higher computation burden on the user interface.

Parent Tags

Parent Tag	Comment
`param`	This is the only valid tag for the `<helptext>` tag. The text specified by this tag will be associated with the particular parameter associated with this parent tag.

Child Tags

Child Tag	Comment
`HTML TAGS`	In general HTML tags as well as HTML text can be contained within the `<helptext>` tag. The quality of the display of the HTML text is limited by the of the current built-in Java HTML rendering cabilities.

Attributes

Attribute	Required	Purpose
`title`	yes	The title that will appear along with the helptext when it is displayed.

Example

The following is a simple example of some help text describing how to select the proper enumerated value for the Existing Utilities parameter.

```
<param name="existing-utilities" description="Existing Utilities"
                  type="enum" default="complex">
      <item display="None"    value="none" />
      <item display="Average" value="average" />
      <item display="Complex" value="complex" />
      <helptext title="Existing Utilities"><b>Average</b> -
            Select for most suburban residential
            settings where some minor utilities, such as water
            service lines, may need to be relocated.
            <br/>
            <br/>
            <b>Complex</b> - Select for heavily urbanized settings.
            <br/>
            <br/>
            If there are known utility conflicts, <b>None</b> should
            be selected for this parameter, the cost for utility
            relocation should be computed separately, and then entered
            manually under &quot;unique construction costs&quot;.
      </helptext>
</param>
```

`<icon>` *Tag*

The `<icon>` tag specifies the path to a custom Icon to be displayed by Tabula to represent the associated model item in the Tabula user interface. The icon is typically a 16x16 color image stored in a GIF file. It must be located in the Java classpath because it is loaded using the Java `ClassLoader.getSystemResource` method. The icon appears next to the model name in the insert menu as well as associated with the default editing screen (unless a customized editing screen is specified using the `<ui>` tag).

Parent Tags

Parent Tag	Comment
`domain`	This is the only valid tag that can be a parent of the `<icon>` tag.

Child Tags

The `<icon>` tag contains no child tags.

Attributes

Attribute	Required	Purpose
`src`	yes	The path to the custom icon. This value is passed to the `ClassLoader.getSystemResource` method while attempting to load the icon. Therefore the icon must be placed somewhere that is visible in the Java classpath.

Example

The following example is an example of the icon location for project domain.

```
<domain type="project" description="Project" order="10">
      <ui class="tabula.model.project.ProjectInfoFrame" />
      <icon src="tabula/model/project/tree_icon.gif" />
      <param name="project-year" description="Project Year"
                  type="year" default="2001"/>
      <param name="comments" description="Comments"
                  type="comment" default=""/>
</domain>
```

When Tabula initializes the user interface for the Project model object, it will attempt to load an image file named “`tree_icon.gif`” that is located in the Java package `tabula.model.project`.

`<ifgt>` *Tag*

The `<ifgt>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *ifgt* operation computes the *sum* of the results of immediately contained child operators if the specified reference evaluates to a value greater than the specified threshold; otherwise, it evaluates to zero.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

Attribute	Required	Purpose
`reference`	yes	The name of a previously defined *variable* object (defined using the `<var>` tag) within the scope of the current calculator object. The value of this variable will be compared to the threshold value.
`value`	yes	The threshold value that the computed *variable* value must exceed for this operator to return the sum of the evaluation of its child *opearators*. If the value is not exceeded, the children are not evaluated and this operator evaluates to zero.

Example

The following example shows the calculation of a minimum allowed easement width.

```
<var name="min-easment-width"
            description="Min. Easement Width" units="ft">
      <sum>
            <number value="20"/>
            <ifgt reference="excavation-depth" value="12">
                  <number value="10"/>
            </ifgt>
            <ifgt reference="excavation-depth" value="20">
                  <number value="20"/>
            </ifgt>
      </sum>
</var>
```

The minimum easement is 20 feet. However if the excavation depth exceeds 12 feet, an additional 10 feet of easement is added. In the event that the excavation depth exceeds 20 feet, the easement width is incremented another 20 feet.

`<ifle>` *Tag*

The `<ifle>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *ifle* operation computes the *sum* of the results of immediately contained child operators if the specified reference evaluates to a value less than or equal to the specified threshold; otherwise, it evaluates to zero.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

Attribute	Required	Purpose
`reference`	yes	The name of a previously defined *variable* object (defined using the `<var>` tag) within the scope of the current calculator object. The value of this variable will be compared to the threshold value.
`value`	yes	The threshold value that the computed *variable* value must be equal to or less than for this operator to return the sum of the evaluation of its child *opearators*. If the value exceeds this threshold, the children are not evaluated and this operator evaluates to zero.

Example

The following example shows the calculation of an easement width.

```
<var name="min-easment-width"
            description="Min. Easement Width" units="ft">
      <sum>
            <number value="40"/>
            <ifle reference="excavation-depth" value="20">
                  <number value="-20"/>
            </ifle>
            <ifle reference="excavation-depth" value="12">
                  <number value="-10"/>
            </ifle>
      </sum>
</var>
```

The base easement value is 40 feet. However if the excavation is less than 20 feet deep, 20 feet are removed from the requirement. In the event that the excavation does not even go as deep as 12 feet, the easement width requirement is decremented another 10 feet.

`<index>` *Tag*

The `<index>` tag specifies individual ENRCCI values. It can appear as many times as necessary inside the `<enrcci>` tag to specify all know ENRCCI values. If multiple values for a given year are specified (either in the same `<enrcci>` tag or spanning across multiple tags) the latest specification in document order will be the one used by Tabula. This tag can appear in both configuration and data files, however ENRCCI values specified in the data file are subordinate to the ones defined in the configuration file.

Parent Tags

Parent Tag	Comment
`enrcci`	This is the only valid parent for the `<index>` tag.

Child Tags

The `<index> tag contains no children tags.`

Attributes

Attribute	Required	Purpose
`year`	yes	The year associated with this ENRCCI value.
`value`	no	The ENRCCI value for the specified year.

Example

An example of the `<index>` tag follows:

```
<enrcci inflation="0.03" >
      <index year="1970" value="1408.2"/>
      <index year="1971" value="1569.78"/>
      <index year="1972" value="1679.62"/>
</enrcci>
```

This `<enrcci>` tag contains three ENRCCI values for 1970, 1971 and 1972 of 1408.2, 1569.78 and 1679.62 respectively. The values do not need to be in any specific order. If multiple values are specified for the same year, the value that appeared last in the document order will be used.

`<inverse>` *Tag*

The `<inverse>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *inverse* operation computes the inverse of the *sum* of the results of immediately contained child operators. In the event that the sum is zero, this operator evaluates to "infinity".

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example shows the conversion from inches to feet by dividing by 12 using the inverse operator.

```
<var name="outer-diameter" description="Outer Diameter"
                              units="ft" category="Geometry">
      <multiply>
            <lookup model-param="diameter" xref="concrete-odiam" />
            <inverse><number value="12"/></inverse>
      </multiply>
</var>
```

`<item>` *Tag*

The `<item>` tag enumerates specific valid entries for enumerated parameters of a model object (described by the `<param>` tag with `type` set to `enum`) or line entries in a cross-reference table (described by the `<xref>` tag.)

Parent Tags

Parent Tag	Comment
`param`	Defines variable parameters that must be defined by user input for a model type. For `<item>` sub-tags to be useful, the type attribute of the `<param>` tag must be set to "`enum`".
`xref`	Associates pairs of values in a lookup table such as linear cost of concrete pipe as a function of its diameter. The associated calculators can then use this lookup table through the `<lookup>` *operator* tag.

Child Tags

The `<item>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`display`	`<param>`	Required when a child of the `<param> tag. This is the display name that Tabula will present to the user in drop-down lists of enumerated values.`
`value`	yes	The actual numerical or programmatic value of the enumerated item. This is the value that is referenced by associated calculators and *operators*.
`result`	`<xref>`	Required when a child of the `<xref>` tag. This is the value returned by the *lookup* operator if it finds a match with the `value` attribute of this cross-reference item.

Example

The following example illustrates the use of the `<item>` tag to define the three types of a pipe:

```
<param name="type" description="Conduit Type"
                              type="enum" default="gravity">
      <item display="Gravity Sewer"             value="gravity"/>
      <item display="Force Main"                value="force"/>
      <item display="High Head Force Main"      value="high-force"/>
</param>
```

While the user will see options such as “Gravity Sewer” the actual value used internally for computations will be values such as “gravity” or “force”. This discretely enumerated value is usually referenced by the `<when>` *operator* in calculations.

When used in conjunction with the `<xref>` tag:

```
<xref name="outside-diameter" value-units="ft" result-units="ft" >
      <item value="8" result="9.25"/>
      <item value="9" result="10.25"/>
      <item value="10" result="11.33"/>
      <item value="11" result="12.33"/>
      <item value="12" result="13.33"/>
      <item value="13" result="14.33"/>
      <item value="14" result="15.5"/>
      <item value="15" result="16.5"/>
      <item value="16" result="17.5"/>
      <item value="18" result="19.5"/>
</xref>
```

The example shows the mapping between the inside diameter and outside diameters of a pipe. This mapping is utilized through use of the `<lookup>` *operator*.

`<lookup>` *Tag*

The `<lookup>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *lookup* operation performs a cross-reference lookup with values defined using the `<xref>` tag for the associated model domain object. If a match is not found the operator evaluates to NaN.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

The `<lookup>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`model-param`	yes	The value of the model parameter (defined by the `<param>` tag) with this name is retrieved from the associated model and then compared in the lookup table for a match.
`xref`	yes	The name of the cross-reference table containing the list of value pairs to search in.

Example

The following example shows the calculation of the outer pipe diameter.

```
<var name="outer-diameter" description="Outer Diameter"
                              units="ft" category="Geometry">
      <multiply>
            <lookup model-param="diameter" xref="concrete-odiam" />
            <inverse><number value="12"/></inverse>
      </multiply>
</var>
```

There is no particular mathematical formula for converting inner pipe diameters to industry standard outer diameter dimensions; therefore it is appropriate to use a cross-reference lookup.

`<max>` *Tag*

The `<max>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *max* operation evaluates to the largest value of the evaluations of its immediately contained child operators.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example shows the computation of the excavation depth for a parallel pipe:

```
<var name="excavation-depth" description="Excavation Depth"
                                  units="ft" category="Geometry">
      <sum>
            <modelparam name="cover-depth"/>
            <max>
                  <reference name="outer-diameter"/>
                  <reference name="outer-diameter2"/>
            </max>
            <number value="1"/>
      </sum>
</var>
```

The maximum diameter of the two pipes in parallel is required for the proper computation of the excavation depth.

`<min>` *Tag*

The `<min>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *min* operation evaluates to the smallest value of the evaluations of its immediately contained child operators.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example shows the computation of the minimum pavement restoration width:

```
<var name="min-pavement-width"
                  description="Complete Surface Rest. Width"
                  units="ft" category="Geometry">
      <sum>
            <when model-param="pavement-restoration" equals="trench">
                  <sum>
                        <reference name="trench-width"/>
                        <number value="2"/>
                  </sum>
            </when>
            <min>
                  <sum>
                        <reference name="trench-width"/>
                        <number value="2"/>
                  </sum>
                  <lookup model-param="pavement-restoration"
                                    xref="pavement-restoration-width"/>
            </min>
      </sum>
</var>
```

In this example, the minimum restoration width is either trench width itself (if the trench is narrower than the pavement width) or the entire width of the pavement itself (if the trench is wider than the pavement).

`<modelconst>` *Tag*

The *modelconst* operation references a constant value specified in the domain definition of the associated model (defined using the `<const>` tag). It is typically used to reference unit cost values.

The `<modelconst>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

The `<modelconst>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`name`	yes	The constant parameter name specified in the `name` attribute of the `<const>` tag.

Example

The following example shows a typical usage of this tag:

```
<cost name="spoils-haul-cost" description="Spoils Haul">
      <quantity units="CY">
            <reference name="tunnel-spoils"/>
      </quantity>
      <unitcost>
            <modelconst name="spoil-haul-unitcost"/>
      </unitcost>
</cost>
```

The unit cost for hauling spoils is not a user-configurable parameter and therefore it is appropriate that it is a constant. It would be possible to use the `<number>` *operator* and include the value of the number directly; but templates and inheritance can be incorporated and overall readability of the configuration file is improved by using constants.

<modelparam> *Tag*

The *modelparam* operation references the value of a model object parameter (defined using the <param> tag).

The <modelparam> tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
quantity	Defines the calculations for computing the quantity value for a cost item.
unitcost	Defines the calculations for computing the unit cost value for a cost item.
var	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
year	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
multiplier	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a <quantity>, <unitcost>, <var> or <year> tag) is calculated.

Child Tags

The <modelconst> tag contains no children tags.

Attributes

Attribute	Required	Purpose
name	yes	The name of the model parameter to reference.

Example

The following example shows a typical usage of this tag:

```
<var name="excavation-depth" description="Excavation Depth"
                              units="ft" category="Geometry">
      <sum>
            <modelparam name="cover-depth"/>
            <reference name="outer-diameter"/>
            <number value="1"/>
      </sum>
</var>
```

The excavation depth is calculated from both the cover depth specified by the user and the computed outer diameter value.

`<multiplier>` *Tag*

The `<multiplier> tag` specifies an additional multiplier to be applied to the cost subtotal of unit costs. This is in addition to the multiplier applied for conversion of project costs to different year basis. One such example is adding 10 percent for mobilization and demobilization of equipment. It only exists as a child tag of the `<calculator>` tag.

Parent Tags

Parent Tag	Comment
`calculator`	The only valid parent tag for this tag.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	The `<multiplier>` parent tag can contain only one *operator* tag. If multiple tags appear as children, only the last one defined will be used. The *operator* tags define the computation of the scaling factor to be applied to the unitcost subtotal of a model item.

Attributes

Attribute	Required	Purpose
`name`	yes	The programmatic name of the multiplier. If the name matches a previously defined multiplier name from an inherited template, this definition replaces the previous definition.
`description`	yes	The description of the multiplier that is displayed in detailed output reports.

Example

This simple example shows the constant 10 percent adjustment for mobilization and demobilization of equipment:

```
<multiplier name="mobilization-demobilization"
                  description="Mobilization/Demobilization at 10%">
      <number value="1.1"/>
```

```
</multiplier>
```

`<multiply>` *Tag*

The *multiply* operation evaluates to the value of the product of the individual evaluations of its immediately contained child operators.

The `<multiply>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example shows the use of the `<multiply>` tag for computing the trench width:

```
<var name="trench-width" description="Trench Width"
                                  units="ft" category="Geometry">
      <sum>
            <multiply>
                  <number value="1.3"/>
                  <reference name="outer-diameter"/>
            </multiply>
            <number value="2.5"/>
      </sum>
</var>
```

`<number>` *Tag*

The *number* operation evaluates to a specified numerical value.

The `<number>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

The `<number>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`value`	yes	The numeric value that this operator should return when it is evaluated.

Example

The following example shows the use of the `<number>` tag for computing the trench width:

```
<var name="trench-width" description="Trench Width"
                                  units="ft" category="Geometry">
      <sum>
            <multiply>
                  <number value="1.3"/>
                  <reference name="outer-diameter"/>
            </multiply>
            <number value="2.5"/>
      </sum>
</var>
```

`<param>` *Tag*

The `<param>` tag exists in two contexts: as a child of the `<domain>` tag in a configuration file and as a child tag of the `<data>` tag in a data file. For information on the data file details, please se the `<param>` tag definition in the data file format section.

The `<param>` tag defines variable parameters that must be defined by user input for the enclosing model domain definition. The values for these parameters for a given model instance are specified by using the `<param>` tag inside a `<data>` tag inside the associated `<model>` tag in a data file.

Parent Tags

Parent Tag	Comment
`domain`	Describes the properties and constants associated with a specific model type.

Child Tags

Child Tag	Comment
`item`	Enumerates specific valid entries for enumerated parameters (if the type of the enclosing parameter equals `"enum"`).
`helptext`	An optional tag that specifies context sensitive help text to be displayed by Tabula when the user focus is on the corresponding editing field for this parameter in a model editing screen.
`range`	Optional, specifies a valid range of values for numerical parameter types.

Attributes

Attribute	Required	Purpose
`name`	yes	Programatic name of the parameter used by calculators to reference values (`<modelparam>` tag). If the name is the same as a name of a parameter defined in a inherited template, this definition replaces the previous definition.
`type`	yes	The type of parameter can be one of: `length`, `volume`, `area`, `rate`, `decimal`, `cost`, `year`, `percent`, `comment`, `boolean` or `enum`.
`description`	yes	A description of the parameter that will be displayed on editing screens and detailed output reports.
`units`	no	A textual notation of the units for this parameter.
`default`	yes	The initial value for this parameter when creating new model objects or to be used when the user does not specify the value for this parameter.

Example

The following example is an example of the defining the length parameter of a pipe:

```
<param name="length" description="Length"
                              type="length" units="ft" default="100"/>
```

`<power>` *Tag*

The *power* operation evaluates to summation taken to the power of the specified exponent of the evaluations of its immediately contained child operators.

The `<power>` tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

Attribute	Required	Purpose
`exponent`	yes	The value of the exponent.

Example

The following example shows the usage of the `<power>` tag.

```
<var name="base-architectural-unitcost"
            description="Base Architectural/Structural Unit Cost"
            units="$/mgd" category="Calculated Parameters">
      <multiply>
            <number value="0.42"/>
            <power exponent="-0.47">
                  <modelparam name="firm-capacity"/>
            </power>
            <number value="1000000"/>
      </multiply>
</var>
```

`<quantity>` *Tag*

The `<quantity>` tag specifies the calculations to determine the quantity portion of a line item costs. It is found inside the `<cost>` tag.

Parent Tags

Parent Tag	Comment
`cost`	Defines the calculations that compute an individual line item cost for a model object and is one of the three types of children tags of the `<calculator>` tag.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside this tag. Only one operator tag can appear as the immediate child of the `<quantity>` tag. If there are multiple tags, the last one defined will replace all other definitions.

Attributes

Attribute	Required	Purpose
`units`	yes	Units designation that appears on output reports.

Example

The following shows an example of computing the excavation cost of an item:

```
<cost name="excavation-cost" description="Excavation">
      <quantity units="CY">
            <multiply>
                  <reference name="trench-width"/>
                  <reference name="excavation-depth"/>
                  <modelparam name="length"/>
                  <inverse><number value="27"/></inverse>
            </multiply>
      </quantity>
      <unitcost>
            <modelconst name="excavation-unitcost"/>
      </unitcost>
</cost>
```

`<range>` *Tag*

The `<range>` tag adds minimum and maximum value constraints to the associated numerical parameter.

Parent Tags

Parent Tag	Comment
`param`	Defines variable parameters that must be defined by user input for a model type. The `<range>` constraint may not apply to all parameter types.

Child Tags

The `<param>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`min`	`no`	Sets the inclusive minimum allowed value for the parameter.
`max`	no	Sets the inclusive maximum allowed value for the parameter.

Example

The following example shows how user input can be restricted to reasonable values for the depth of cover:

```
<param name="cover-depth" description="Depth of Cover"
                        type="decimal" units="ft" default="12">
      <range min="4" max="40"/>
</param>
```

`<reference>` *Tag*

The `<reference>` tag is a mathematical *operator* tag. The *reference* operation references the value of a previously computed intermediate variable (defined by the `<var>` tag) residing within the same calculator context. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

The `<reference>` tag contains no children tags.

Attributes

Attribute	Required	Purpose
`name`	yes	Name of the intermediate variable to reference.

Example

The following example shows the computation of the excavation depth for a parallel pipe:

```
<var name="excavation-depth" description="Excavation Depth"
                                  units="ft" category="Geometry">
      <sum>
            <modelparam name="cover-depth"/>
            <max>
                  <reference name="outer-diameter"/>
                  <reference name="outer-diameter2"/>
            </max>
            <number value="1"/>
      </sum>
</var>
```

The values for the outer diameters for the parallel pipes are needed for many calculations. It makes sense to explicitly define them as intermediate variables. They are then referenced to compute the excavation depth.

`<sum>` *Tag*

The `<sum>` tag is a mathematical *operator* tag. The *sum* operation evaluates to the sum of the evaluations of its immediately contained child operators. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a `<quantity>`, `<unitcost>`, `<var>` or `<year>` tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

This tag contains no attributes.

Example

The following example shows the computation of the excavation depth for a parallel pipe:

```
<var name="excavation-depth" description="Excavation Depth"
                                    units="ft" category="Geometry">
      <sum>
            <modelparam name="cover-depth"/>
            <max>
                  <reference name="outer-diameter"/>
                  <reference name="outer-diameter2"/>
            </max>
            <number value="1"/>
      </sum>
</var>
```

The excavation depth is the *sum* of the cover depth, largest outer pipe diameter plus one foot extra.

`<tabula>` *Tag*

The `<tabula>` tag is the root of all Tabula Files, both configuration files and data files. For details on data file requirements, please see the data file reference. In configuration files, the `<tabula>` tag typically contains `<enrcci>`, `<calculator>` and `<domain>` child tags.

Parent Tags

This is the root tag of Tabula files; there are no parent tags.

Child Tags

Child Tag	Comment
`enrcci`	Represents Cost Escalation Factors. This tag can specify both Seattle ENRCCI values as well as the inflation rate where ENRCCI values are not known.
`calculator`	This tag and its enclosed tags define the calculations used to compute the estimated cost of an individual model object. The calculator definition can inherit instructions from a previously defined calculator and act as a *template* for calculators yet to be defined.
`domain`	Describes the properties and constants associated with a model object. It provides the definition for input parameters, their types and limitations.

Attributes

Attribute	Required	Purpose

`<ui>` *Tag*

The `<ui>` tag specifies a Java Class Object that provides a customized editing screen for the associated model object. The Java Class Object must extend `javax.swing.JinternalFrame` and provide a public constructor that takes a `tabula.ui.ModelRelay` and `tabula.ui.ViewManager` as arguments. This is the way the Tabula user interface is notified of the existence of custom editing screens. Implementation of this feature requires a working knowledge of the Java source code for Tabula. The Java class referenced by the `<ui>` tag must be visible in the Java classpath.

Parent Tags

Parent Tag	Comment
`domain`	This is the only valid tag that can be a parent of the `<ui>` tag.

Child Tags

The `<ui>` tag contains no child tags.

Attributes

Attribute	Required	Purpose
`class`	yes	The fully qualified class name of the custom editing screen.

Example

The following example is an example of the user interface designation for the project domain.

```
<domain type="project" description="Project" order="10">
      <ui class="tabula.model.project.ProjectInfoFrame" />
      <icon src="tabula/model/project/tree_icon.gif" />
      <param name="project-year" description="Project Year"
                  type="year" default="2006"/>
      <param name="comments" description="Comments"
                  type="comment" default=""/>
</domain>
```

When Tabula initializes the user interface for the Project model object, it will attempt to load the class `tabula.model.project.ProjectInfoFrame`. If for some reason it is unable to complete the loading process of this Java class, it will display a message stating that it cannot load the editing screen for this model.

`<unitcost>` *Tag*

The `<unitcost>` tag contains a child *operator* tag that defines the instructions for computing the unit cost value for the associated cost line item. It may have only one child *operator* tag, if it contains multiple child tags, then the last one defined will overwrite the previous definitions.

Parent Tags

Parent Tag	Comment
`cost`	Defines the calculations that compute an individual line item cost for a model object and is one of the three types of children tags of the `<calculator>` tag. A cost item has both a unit cost (defined by this tag) and number of units required, represented by the `<quantity>` tag.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside this tag. Only one operator tag can appear as the immediate child of the `<unitcost>` tag. If there are multiple tags, the last one defined will replace all other definitions.

Attributes

The `<unitcost>` tag contains no attributes.

Example

The following illustarates the use of the `<unitcost>` tag:

```
<cost name="ls-excavation-cost" description="Launch Shaft Excavation">
      <quantity units="CY">
            <reference name="ls-volume"/>
      </quantity>
      <unitcost>
            <modelconst name="shaft-excavation-unitcost"/>
      </unitcost>
</cost>
```

`<var>` *Tag*

The `<var>` tag defines intermediate numerical computational variables that can be reused by other calculations and can appear on detailed output reports. The calculated values can be referenced in other calculations via use of the `<reference>` tag.

Parent Tags

Parent Tag	Comment
`calculator`	The only valid parent tag for this tag.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	The `<var>` parent tag can contain only one *operator* tag. If multiple tags appear as children, only the last one defined will be used. The *operator* tags define the computation of the scaling factor to be applied to the unitcost subtotal of a model item.

Attributes

Attribute	Required	Purpose
`name`	yes	The programmatic name of the variable. If the name matches a previously defined variable name from an inherited template, this definition replaces the previous definition.
`description`	yes	The description of the variable that is displayed in detailed output reports.
`units`	no	Specifies units for the variable (if necessary.)
`category`	no	Designates which category this variable should appear under in a detailed output report. If none is specified, this variable will not be displayed on detailed output reports.

Example

The following example shows the calculation of a minimum allowed easement width.

```
<var name="min-easment-width"
            description="Min. Easement Width" units="ft">
      <sum>
            <number value="20"/>
            <ifgt reference="excavation-depth" value="12">
                  <number value="10"/>
            </ifgt>
            <ifgt reference="excavation-depth" value="20">
                  <number value="20"/>
            </ifgt>
      </sum>
</var>
```

<when> *Tag*

The <when> tag is a mathematical *operator* tag. It appears in places that define the mathematical computations necessary to estimate the cost of the model. It can be used in the context of a variable, cost or year calculation. It can be an immediate child of the variable, unit cost, unit price and year objects as well as a child of another operation tag.

The *when* operation evaluates to sum of the evaluations of its immediately contained child operators *if* the where condition is satisfied, else it evaluates to zero. The where condition is satisfied if the value of the model parameter specified by the `model-param` attribute matches the value specified in the `equals` attribute.

Parent Tags

Parent Tag	Comment
`quantity`	Defines the calculations for computing the quantity value for a cost item.
`unitcost`	Defines the calculations for computing the unit cost value for a cost item.
`var`	Defines the calculations for computing the value of an intermediate variable that is subsequently used in calculations for a cost item.
`year`	Defines the calculations for computing the construction year value to be used in cost escalation calculations.
`multiplier`	Tag indicating that the contained operator and child operators define the calculations for computing an additional multiplier to be applied to the cost subtotal of unit costs.
`ceiling, floor, ifgt, ifle, inverse, max, min, multiply, power, sum, when`	Operator tags that can contain this and other operator tags. Nesting of operator tags form a *tree* of operations. The values of the leaf operators of the tree are computed first, followed by the next level of enclosing operators until the root operator (the operator tag that is the immediate child of a <quantity>, <unitcost>, <var> or <year> tag) is calculated.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	Operator tags that can appear inside the context of this operator tag.

Attributes

Attribute	Required	Purpose
`model-param`	yes	The value of the model parameter with this name is compared to the value specified in the `equals` attribute to determine if the *where condition* is satisfied.
`equals`	yes	The value that the model parameter value must match in order to satisfy the *where condition*.

Example

The following example shows how a cost calculation can adapt to two different types of dewatering conditions by using the `<when>` tag.

```
<cost name="dewatering-cost" description="Dewatering">
      <quantity units="lf">
            <modelparam name="length"/>
      </quantity>
      <unitcost>
            <sum>
                  <when model-param="dewatering" equals="minimal">
                        <lookup model-param="diameter"
                              xref="trench-sump-dewatering-unitcost"/>
                  </when>
                  <when model-param="dewatering" equals="significant">
                        <lookup model-param="diameter"
                              xref="dewatering-well-unitcost" />
                  </when>
            </sum>
      </unitcost>
</cost>
```

`<xref>` *Tag*

The `<xref>` tag is enclosed inside a `<calculator>` tag. It defines a lookup table that can be used to map value pairs to each other. It is referenced in calculations by the `<lookup>` *operator* tag.

A definition of a cross-reference table by using the `<xref>` tag overrides any inherited cross-reference definitions defined in any parent templates used by the model object calculator description.

Parent Tags

Parent Tag	Comment
`calculator`	This is the only parent for the `<xref>` tag.

Child Tags

Child Tag	Comment
`item`	Specifies an individual value pair entry for the table.

Attributes

Attribute	Required	Purpose
`name`	yes	A unique programmatic name of the cross-reference table. If this name matches the name of a previously defined cross-reference table in a parent template, this definition will override the previous definition.
`value-units`	no	Units specifier for the lookup value.
`result-units`	no	Units specifier for the result value.

Example

The following illustrates a use of the `<xref>` tag to establish a cross-reference table.

```
<xref name="pavement-restoration-width"
                              value-units="enum" result-units="ft">
      <item value="none"              result="0" />
      <item value="trench"            result="0" />
      <item value="half"              result="22" />
      <item value="full-collector"    result="18" />
      <item value="full-residential"  result="14" />
      <item value="full"              result="44" />
      <item value="full-collector"    result="36" />
      <item value="full-residential"  result="28" />
</xref>
```

The table associates descriptions of restoration widths selected by the user with the actual numerical values.

`<year>` *Tag*

The `<year>` tag specifies the calculations for computing the construction year for the associated model object. It only exists as a child tag of the `<calculator>` tag.

Parent Tags

Parent Tag	Comment
`calculator`	The only valid parent tag for this tag.

Child Tags

Child Tag	Comment
`ceiling, modelconst, floor, igt, ifle, inverse, lookup, min, max, multiply, number, modelparam, power, reference, sum, when`	The `<year>` parent tag can contain only one *operator* tag. If multiple tags appear as children, only the last one defined will be used.

Attributes

The `<year>` tag contains no attributes.

Example

This simple example shows the usage of the `<year>` tag in the calculator definition for the `project` type.

```
<calculator type="project" description="Project" enrcci-year="1999"
            comment="The estimated construction cost below, which
            includes contractor overhead and profit,  is for planning purposes only.
            The output does NOT include contingency, sales tax, or allied costs
            (design, permitting, construction management, etc. )."">
      <year>
            <modelparam name="project-year"/>
      </year>
</calculator>
```

Data File Reference

This section provides descriptions of the tags most commonly found in Tabula project files (sometimes called *data* files). The data file describing a Tabula project appears much like the configuration file. In fact, most components tags that can be contained in a configuration file can also exist inside a project data file.

Domains and *Calculators* can exist as children of `<model>` tag to define custom sub domains and calculators for specialized models. These capabilities are available for users that wish to create customized objects that are not included in the default Tabula installation. This functionality was included in the original design of the system to allow for future expansion of the program but is not utilized at this time and is therefore not fully documented.

<data> *Tag*

The <data> tag denotes the section of a <model> tag where the user-configured properties are stored. It contains one or many <param> tags that associate the name of the parameter with its user specified value. This was done for readability; it might very well prove difficult to read a file where sub-model tags are interspersed with parameter value tags.

Parent Tags

Parent Tag	Comment
model	The <model> tag is the only parent tag for this tag.

Child Tags

Child Tag	Comment
param	Associates the user specified value with a model object parameter.
item	Specifies a custom “unique cost” that was entered by the end user. There is one <item> tag for each and every unique cost entered by the user.

Attributes

This tag has no attributes.

Example

The following example is an example of a model that contains no sub-models and has two custom cost items.

```
<model type="pipe" name="Pipe One">
      <data>
            <param name="construction-year" value="2003"/>
            <param name="length" value="100"/>
            <param name="type" value="gravity"/>
            <item name="Item &lt; number 1"
                        quantity="1.0" units="LS" unitcost="1000.0"/>
            <item name="Item &gt; number 2" quantity="500.0"
                                          units="sf" unitcost="2.0"/>
      </data>
</model>
```

`<enrcci>` *Tag*

The `<enrcci>` tag represents Cost Escalation Factors. It can specify both Seattle ENRCCI values as well as the inflation rate where ENRCCI values are not known. The tag is typically the first child of the `<tabula>` tag (in both configuration and data files), but can appear anywhere as a child of the `<tabula>` tag. It will be ignored if appearing as a child of any other tag. If multiple values for a given year are specified (either in the same `<enrcci>` tag or spanning across multiple tags) the latest specification in document order will be the one used by Tabula.

Parent Tags

Parent Tag	Comment
`tabula`	This is the only valid parent for the `<enrcci>` tag.

Child Tags

Child Tag	Comment
`index`	Specifies a particular ENRCCI value entry for a given year.

Attributes

Attribute	Required	Purpose
`inflation`	yes	The numerical value for the inflation rate to be applied to calculations where ENRCCI values are not known. This value is specified decimal value, not percentage (unlike the Tabula UI itself, which requires the value to be specified as a percentage).
`roundcalc`	no	Specifies that number values should be rounded to three significant figures between calculations. Specify the value `"true"` for turning this feature on and `"false"` for turning it off. The default is `"false"`. If specified, this value overrides default user settings.
`rounddisplay`	no	Specifies that numerical results should be rounded to three significant figures for display purposes. Specify the value `"true"` for turning this feature on and `"false"` for turning it off. The default is `"false"`. If specified, this value overrides default user settings.

Example

An example of the `<enrcci>` tag follows:

```
<enrcci inflation="0.03" >
      <index year="1970" value="1408.2"/>
      <index year="1971" value="1569.78"/>
      <index year="1972" value="1679.62"/>
</enrcci>
```

The inflation rate (expressed as a decimal value, not a percentage) is specified in the `inflation` attribute of the `<enrcci>` tag. The `<enrcci>` tag may contain one or many `<index>` tags. Each `<index>` tag represents an ENRCCI value for the specified year. The `year` attribute designates the year and the `value` attribute specifies the ENRCCI value. The values do not need to be in any specific order. However, if multiple values are specified for the same year, the value that appeared last in the document order will be used.

`<index>` *Tag*

The `<index>` tag specifies individual ENRCCI values. It can appear as many times as necessary inside the `<enrcci>` tag to specify all know ENRCCI values. If multiple values for a given year are specified (either in the same `<enrcci>` tag or spanning across multiple tags) the latest specification in document order will be the one used by Tabula. This tag can appear in both configuration and data files, however ENRCCI values specified in the data file are subordinate to the ones defined in the configuration file.

Parent Tags

Parent Tag	Comment
`enrcci`	This is the only valid parent for the `<index>` tag.

Child Tags

The `<index> tag contains no children tags.`

Attributes

Attribute	Required	Purpose
`year`	yes	The year associated with this ENRCCI value.
`value`	no	The ENRCCI value for the specified year.

Example

An example of the `<index>` tag follows:

```
<enrcci inflation="0.03" >
      <index year="1970" value="1408.2"/>
      <index year="1971" value="1569.78"/>
      <index year="1972" value="1679.62"/>
</enrcci>
```

This `<enrcci>` tag contains three ENRCCI values for 1970, 1971 and 1972 of 1408.2, 1569.78 and 1679.62 respectively. The values do not need to be in any specific order. If multiple values are specified for the same year, the value that appeared last in the document order will be used.

`<item>` *Tag*

The `<item>` tag represents user entered unique construction costs for the given model object. There is one `<item>` tag for each and every unique construction cost.

Parent Tags

Parent Tag	Comment
`data`	This is the only parent tag for this tag in data files.

Child Tags

This tag contains no child tags.

Attributes

Attribute	Required	Purpose
`name`	yes	Description of the unique construction cost item or procedure.
`quantity`	yes	Quantity of the unique item or procedure.
`units`	yes	Units descriptor for the unique item or procedure.
`unitcost`	yes	The cost per unit of unique item or procedure.

Example

The following example is an example of a model that contains no sub-models and has two custom cost items.

```
<model type="pipe" name="Pipe One">
      <data>
            <param name="construction-year" value="2003"/>
            <param name="length" value="100"/>
            <param name="type" value="gravity"/>
            <item name="Item &lt; number 1"
                        quantity="1.0" units="LS" unitcost="1000.0"/>
            <item name="Item &gt; number 2" quantity="500.0"
                                    units="sf" unitcost="2.0"/>
      </data>
</model>
```

`<model>` *Tag*

The `<model>` tag is the most important tag in the project data file. The tag represents an instance of a model object of a specific type (such as a pipe or pump station). Models can contain sub-models (which is specified by placing a `<model>` tag as a child of an existing `<model>` tag). A `<model>` tag can also contain `<calculator>` and `<domain>` sub-tags, however this advanced functionality is not covered in this document.

Parent Tags

Parent Tag	Comment
`tabula`	This is the root tag of the data file. All tabula files have this tag as the root tag.
`model`	A parent model, this model represents a sub-model of the enclosing model.

Child Tags

Child Tag	Comment
`model`	Specifies a sub-model to be contained within this model.
`data`	Delineates the data section that specifies the type-specific parameters associated with this instance of a model object.

Attributes

Attribute	Required	Purpose
`type`	yes	The type name of the model. It the type name must match a type name of an defined domain within the same scope that contains this model object.
`name`	yes	User specified name for the model. The name does not need to be unique, however doing so makes the model easier to read and comprehend.

Example

An example of the model of type "`pipe`" follows:

```
<model type="pipe" name="Mainline">
    <data>
        <param name="construction-year" value="2006" />
        <param name="length" value="100" />
```

```
            <param name="type" value="gravity" />
            <param name="cover-depth" value="12" />
            <param name="backfill" value="imported" />
            <param name="disposal" value="none" />
            <param name="manhole-spacing" value="500" />
            <param name="existing-utilities" value="complex" />
            <param name="dewatering" value="minimal" />
            <param name="pavement-restoration" value="trench" />
            <param name="traffic" value="heavy" />
            <param name="row" value="none" />
            <param name="easment" value="none" />
            <param name="trench-safety" value="standard" />
            <param name="diameter" value="12" />
        </data>
    </model>
```

This model object is of the type “pipe” (specified by the `type` attribute) with the name “Mainline” (specified by the `name` attribute). The model tag has a sub-tag for instance data called the `<data>` tag. Inside the `<data>` tag are different `<param>` tags specifying the values of the parameter attributes of the model object as specified by its defining domain.

`<param>` *Tag*

The `<param>` tag exists in two contexts: as a child of the `<domain>` tag in a configuration file and as a child tag of the `<data>` tag in a data file. For information on the configuration file details, please see the `<param>` tag definition in the configuration file format section.

In the context of a data file, the `<param> tag records the specific parameter values for instances of model objects. These tags are contained within the <data>` sub-tag of the enclosing `<model>` tag.

If a `<param>` tag is missing for a particular model attribute, Tabula will use the default specified in the corresponding `<domain>` tag. Specifying a value (`value` attribute of the `<param>` tag) that is outside the allowed domain of values for that particular attribute will cause an error when Tabula tries to load the file

Parent Tags

Parent Tag	Comment
`data`	The only parent for the `<param>` tag in the data file context.

Child Tags

This tag contains no child tags.

Attributes

Attribute	Required	Purpose
`name`	yes	Programatic name of the model object attribute defined by its associated domain.
`value`	yes	Value of the model object attribute that was specified by the user.

Example

An example of the parameters for a pipe object follows:

```
<model type="pipe" name="Mainline">
      <data>
            <param name="construction-year" value="2002"/>
            <param name="length" value="1000"/>
            <param name="type" value="gravity"/>
            <param name="cover-depth" value="6"/>
            <param name="backfill" value="imported"/>
            <param name="manhole-spacing" value="250"/>
            <param name="existing-utilities" value="average"/>
            <param name="dewatering" value="none"/>
            <param name="pavement-restoration" value="none"/>
            <param name="traffic" value="none"/>
            <param name="row" value="none"/>
            <param name="easment" value="none"/>
            <param name="trench-safety" value="standard"/>
            <param name="diameter" value="12"/>
      </data>
</model>
```

`<tabula>` *Tag*

The `<tabula>` tag is the root of all Tabula Files, both configuration files and data files. For details on configuration file requirements, please see the configuration file reference. It is convention (and may be required in the future) that data files have a `model` attribute of the `<tabula>` tag set to `"true"`. Typically in data files, the `<tabula>` tag only contains <enrcci> and <model> child tags, but can contain `<calculator>` and `<domain>` tags (advanced functionality).

Parent Tags

This is the root tag of Tabula files; there are no parent tags.

Child Tags

Child Tag	Comment
`enrcci`	Represents Cost Escalation Factors. This tag can specify both Seattle ENRCCI values as well as the inflation rate where ENRCCI values are not known.
`model`	Each project can only have one root model (typically of type "project").

Attributes

Attribute	Required	Purpose
`model`	no	Set to "true" by convention in data files. May be required in future uses.

Appendix A: Source Code Design and Layout

One of objectives of the overall design of the Tabula source code was to produce a calculation core that could easily be extended or extracted from Tabula and then used for other purposes such as batch processing or processing very large files serially. This objective has been achieved in large part. The source code for Tabula divides into five major packages: *tabula.data, tabula.model, tabula.security, tabula.ui,* and *tabula.util*. The calculation core relies principally upon the classes located the *tabula.data,* and *tabula.util* packages. The graphical user interface related code is contained within the *tabula.ui* and *tabula.model* packages. The following sections briefly discuss the roles that each major package plays in the overall Tabula design.

tabula.data. Packages*

The `data` package contains the core definitions of the programmatic representations of a Tabula model. It is where the code implementing such abstract Tabula objects such as Models, Domains and Calculators resides. It also contains other internal objects that help with calculations such as escalating cost values over time spans. There are three sub-packages: *calculator, domain* and *io*. The *calculator* and *domain* packages are simply implementation details used by the Calculator and Domain objects. The *io* package contains the code that reads and writes data and configuration files.

All Tabula data and configuration files are formatted in XML. Tabula uses the Xerces XML library for Java for dealing with most of the XML formatting issues. The library implements a standard called Simple API for XML or SAX for short. The SAX standard requires that client code interfacing with this standard implement classes that provide certain interfaces for the library code to call (as a callback, specifically in this case the *DefaultHandler* interface).

Before loading an XML file into Tabula, the program first creates an instance of the `ModelBuilderContext` class; this object manages the state of the loading process and provides a place where the root Model object can be deposited. Next it creates an instance of the `ModelBuilder` object (which requires an instance of the `ModelBuilderContext` for its construction). This is the class that provides the *DefualtHandler* interface. The loading process then passes this object to the Xerces library and instructs the library to read in the desired XML file. As the library code reads the file it calls certain methods on the `ModelBuilder` class that are then interpreted and translated into calls to the `ModelBuilderContext` that are ultimately used to reconstruct an in-memory model of the XML file that was stored in the file. In most cases, this is repeated twice with the same `ModelBuilderContext` object, the first file being the configuration file to load the domain and calculator definitions and the second file being the desired project data file. After the files are loaded, the program calls methods on the `ModelBuilderContext` to retrieve the root model object and supporting objects.

If sometime in the future there is a requirement to process huge data files, this design pattern could possibly be modified by extending the `ModelBuilderContext` class in such a way that it would process model objects sequentially *in-line* as they are being read (instead of creating an *in-memory* tree representation of the data as it does so now) and

passing on the cost results as they are computed. The major memory requirement would only be a function of the *depth* of the model project tree (how many models are nested inside of other models); this would likely not be a problem in this scenario.

When saving Tabula data into an XML file, the program creates an instance of the `ModelSAXParser` object. This object can translate the contents of the *in-memory* representation of a project into the appropriate calls to the *`DefaultHandler`* interface to reconstruct an XML data file. The program queries Xerces library for an object implementing the *`DefaultHandler`* interface that can translate the SAX calls back into XML stored in a file. It then passes this object to the `ModelSAXParser` object and instructs it to translate the data into SAX calls.

tabula.ui Package

The `ui` package contains the core user interface code for the Tabula program. Auxiliary code for specializing editing screens for various types of model objects are contained in the *tabula.model.** packages.

This package contains a very large number of objects, so it is impossible to provide a review of all of them. However, a few are worth mentioning as they provide the core of the Tabula user interface: the `MainFrame`, `ViewManager`, `ModelRelay` and `ModelManager` objects.

The `MainFrame` object provides the Tabula application window. It stores references to the various user interface widgets required by the program.

The class that manages most of the behavior of the Tabula program is the `ViewManager`. For the most part it manages the resources of the `MainFrame` and responds to user interactions such as selecting items from a menu. To some extent, it also manages the state of various user interface components and keeps the master list of which editing and output windows are displayed and hidden.

The `ModelRelay` object is another crucial component in the system. This object acts as the sole go-between that links user interface components and the underlying project data. This object makes it possible to completely separate the user interface components from the data and calculation components (so that the program can be more easily adapted to for other purposes and environments as stated above). `ModelRelay` objects also maintain the state of the data model; such as if the data has changed since the last time it was written to a file. It knows how to move, delete, and copy model objects from the model tree and other data manipulation routines that are not directly provided by the data objects themselves.

The last of the major UI components is the `ModelManager` object. This object works closely with the `ModelRelay` object to manage the transferring data from the data model out to a file representation and the loading data files into memory. It provides the user interaction for loading and saving data files. It informs the Tabula application of changes in the status of the data by making calls to registered `ModelManagerListener` objects. The `ViewManager` is one of those objects that is interested in the status of the data file.

tabula.model. Packages*

For the most part, Tabula could run adequately without the classes provided in the *tabula.model.** packages; however the user interface would be much blander and not as useful without the customizations provided by the objects in this package.

The packages contained within the *tabula.model* package provide the code to support the specialized editing screens that are customized for each specific model object type. Tabula is made aware of the existence of these classes by usage of the `<ui>` tag in the configuration file, which happens to be located in the *tabula.model* package itself.

Each customized editing screen object extends the `tabula.ui.AbstractInfoFrame` object and provides a constructor that takes a `tabula.ui.ModelRelay` and `tabula.ui.ViewManager` as arguments. The `AbstractInfoFrame` provides many convenience functions that drastically reduce the level of effort required to produce a customized editing screen for a model object. New editing screens are not required to extend the `AbstractInfoFrame` class, however they are required to provide the same constructor signature, implement the `tabula.ui.ModelRelayListener` and eventually extend the `javax.swing.JinternalFrame` class. For more information on the required behavior of custom editing screens please review the `AbstractInfoFrame` and other custom editing screens' source code.

tabula.security Package

The `security` package contains one class, the `SignTabula` class. This is the code that creates the configuration signature file. It is imperative that this file not be accidentally sent out in a distribution of Tabula; doing so would defeat the security measures to prevent tampering with the configuration file.

tabula.util Package

The `util` package contains some generic utility classes that provide convenience functions for dealing with the Xerces XML library.

Appendix B: Configuration File Modification Walkthrough

Modifying the configuration file for Tabula is a relatively straightforward process. Roughly speaking, one needs to obtain a copy, make modifications, generate a signature file, and then replace the old configuration file and signature file with the recently modified files. This appendix walks through this process step-by-step.

Step 1: Obtain a copy of the configuration file.

The easiest place to get a copy of the configuration file is to look in the Tabula application deployment directory. The configuration file is located at:

```
<application_directory>\bin\tabula\model\default_domains.xml
```

where `<application_directory>` is the directory that the Tabula files were installed into (typically "C:\Program Files\Tabula" on Windows). Also, the file can be obtained from the project source code directory:

```
<root>\KingCo-CSI\Projects\tabula\res\tabula\model\default_domains.xml
```

where `<root>` is the development project root directory or drive letter of the CDROM containing Tabula development CD.

Step 2: Obtain a copy of the signing utility

The most current version of the signing utility can be found at:

```
<root>\KingCo-CSI\Projects\tabula\SignTabula.jar
```

where `<root>` is the development project root directory or drive letter of the CDROM containing Tabula development CD.

Step 3: Verify JRE installation

After obtaining copies of the existing configuration file and the signing utility, it is necessary to verify that the current machine has a proper Java runtime installed.

Open up a command prompt (DOS window) and type:

```
java -version
```

The response should look something like:

```
java version "1.3.1"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.3.1-b24)
Java HotSpot(TM) Client VM (build 1.3.1-b24, mixed mode)
```

The first line is the most important one; the version number should read 1.2 or greater.

If the response is something like:

```
'java' is not recognized as an internal or external command,
operable program or batch file.
```

then the Java Runtime Environment (JRE) is not properly installed on the computer. It will be necessary to visit http://www.javasoft.com or a similar site to obtain an installation of the JRE before continuing. Please see this site for installation instructions.

(If a system JRE is not installed on the computer, but Tabula is installed, a workaround is to substitute '`<application_directory>\JRE\1.3.1\bin\java.exe`' where "java" is entered at the command prompt for the previous and following instructions. Remember to use quotes around '`<application_directory>\JRE\1.3.1\bin\java.exe`' if the path contains spaces.)

Step 4: Modify the Configuration File

The next step is to make the desired modifications to the configuration file. This can be done using any text file editor. Remember that this is an XML file so the rules governing the layout of XML files apply. For the details of how to edit and make changes to the configuration file XML tags, please consult the Configuration File Reference above.

Save the changes after finishing the modifications.

Step 5: Creating the *.sig file

Generating the signature file is relatively straightforward. If the JRE version is 1.3 or greater, the command is invoked by typing:

`java -jar SignTabula.jar <configuration_file>`

Where *`<configuration_file>`* is the path to the configuration file to be signed.

If the JRE version is 1.2.x, then the command is slightly modified to the following:

`java -cp SignTabula.jar tabula.security.SignTabula <configuration_file>`

If all goes well, the program should respond with:

```
Reading ... Digesting ... Writing ...Done.
```

Now there should be a file with the same name as the configuration file appended with "`.sig`" in the same directory as the recently edited configuration file. For example if the file was named `default_domains.xml`, there should be an additional file named `default_domains.xml.sig`.

Step 6: Update the Tabula Installation

At this time, no program has been written to deliver just a patch of the configuration file. Therefore updates to the program must be performed manually. The recently edited configuration file and its signature file must be copied back into their default locations in the Tabula's application directory. The configuration file and its signature file should be copied to:

```
<application_directory>\bin\tabula\model\default_domains.xml
<application_directory>\bin\tabula\model\default_domains.xml.sig
```

`respectively. <application_directory>` is the directory that the Tabula files were originally installed into (typically "C:\Program Files\Tabula3" on Windows).

Another option would be to create a new distribution of Tabula; however, this is beyond the scope of this discussion.

Appendix C: Development Environment

This appendix discusses the general layout of the development environment, where files are located and an inventory of the development tools used to create this program. The development of this product was conducted solely on a computer running Windows 2000 Professional.

Development Tools

A number of development tools were utilized to create the complete Tabula program and supporting installation scripts. The following paragraphs discuss the role of each of these tools.

NetBeans 6.5 provided the Integrated Development Environment (IDE) used to write, compile, and debug most of the Java code for the program. NetBeans provides tools for reducing user interface design process from coding to dragging and dropping items on a canvas. This functionality was not used; it was far more practical to write the user interface components from scratch.

Previous versions of Tabula were compiled using JBuilder. Because JBuilder had changed hands twice between 2005 and 2008 and had become sufficiently different from the original version, compiling the existing source code was nearly impossible. In order to simplify future modifications to the code, Sun Microsystems' NetBeans IDE was used to modify and compile the Java code. The NetBeans IDE provides the most generic of Java compilers and will retain backwards compatibility for future revisions of the compiler. It is also readily available from the Sun Microsystems' website and is well supported.

Microsoft Visual Studio 6.0 provided the IDE for developing the bootstrap native Windows launcher program. The launcher program is standard Visual Basic Windows program. The project file for this program resides at \KingCo-CSI\Projects\launcher\Tabula3.vbproj.

Install Creator Version 2.0 Build 32 provided the environment to create the Windows installation program. The project file resides at \KingCo-CSI\Projects\Setup\Tabula3.iit.

Adobe PhotoShop CS2 was used to create and edit graphics for the project.

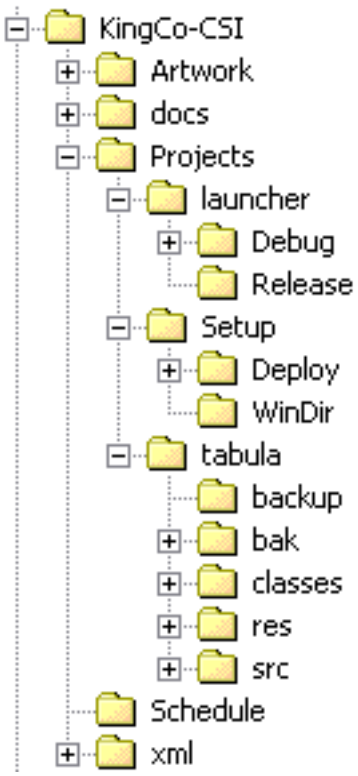
Microsoft Office 2000 was used for various tasks such as writing documents, prototyping calculations etc.

Homesite 5.0 was used to write the HTML the Tabula website download area.

WinZip was used to make compressed backups of previous releases and development efforts.

Development Directory Layout

The following image illustrates the Tabula development directory layout.



The following paragraphs describe the purpose of each directory.

The **Artwork** directory contains the original artwork (typically stored as Paint Shop Pro files) used to create the images, icons and splash screens displayed by Tabula. This is the original source material. Images are copied into other development directories for consumption by the IDE's and compilers.

The **docs** directory and its subdirectories contain development documents and records of pertinent correspondence regarding the development of the program.

The **projects** directory is the main directory containing the source code files and project files for the various sub projects of the entire Tabula programming effort.

The **launcher** project directory contains the all of the bootstrap Windows launcher program source code. The layout of the files and subdirectories of this folder are largely controlled by the Visual Studio IDE. For testing purposes, some JRE runtime files and Tabula class files were copied into the **Debug** subdirectory.

The **Setup** project directory contains the files necessary to create a Tabula installation program. The **WinDir** subdirectory contains a file named `msvcrt.dll` this is the version of the DLL that the resulting installation program will install in the system directory of the target machine if it does not already exist. The **Deploy** directory contains a *complete and most recent* image of the deployment of the application on target computers. When a new version of Tabula is created, the new class files are copied to this location before creating a new installation program. For a discussion of the significance of the layout please see the file directory structure discussion in the Windows Deployment section above.

The **tabula** project directory contains the Java portion of the Tabula project as well as backup copies of previously released versions of Tabula. Inside the **backup** directory lies the previous releases and corresponding source code: files starting with "release" contain copies of the setup script and image for the corresponding release; files starting with "src" contain the source code for the labeled release (appended with a backup number _01, _02 etc.); and files starting with "TabulaSetup" are the installation programs corresponding to the numbered releases. The JBuilder IDE maintains the contents of the **bak** subdirectory, places compiled *class* files in the **classes** directory and maintains the Java source code files in the **src** directory. The **res** directory is where other non-Java-source files that must reside in the Java classpath are kept. (This is much cleaner than keeping them in the **src** directory.) When a new version of Tabula is created the contents of the **class** and **res** directories need to be (selectively) copied to the **Deploy** subdirectory of the **Setup** project directory.

The **Schedule** directory contains old project schedules.

The **xml** directory is a sandbox of XML file fragments used in development and debugging.

Appendix D: ENRCCI Values

Cost escalation calculations use an inflation rate of 4.2 percent when not using ENRCCI values to scale costs. The ENRCCI values as they are currently defined in the Tabula configuration file appear in the table below:

Year	ENR CCI	Year	ENR CCI	Year	ENR CCI
1970	1408.20	1984	4546.01	1998	6957.81
1971	1569.78	1985	4563.10	1999	7137.17
1972	1679.62	1986	4585.40	2000	7368.00
1973	1843.94	1987	4684.28	2001	7335.00
1974	2097.26	1988	4738.35	2002	7562.00
1975	2409.21	1989	4898.01	2003	7867.00
1976	2668.00	1990	4933.39	2004	8165.00
1977	2865.60	1991	5120.63	2005	8390.00
1978	3197.00	1992	5339.69	2006	8641.00
1979	3497.64	1993	5722.73	2007	8618.00
1980	3909.16	1994	5818.49	2008	8815.00
1981	4230.36	1995	5924.09		
1982	4490.38	1996	6086.77		
1983	4559.55	1997	6639.85		

Appendix E: Cost Calculation Atlas

This appendix summarizes the calculation instructions contained in the Tabula XML Configuration file. It lists the details the *Domain* and *Calculator* definitions for each model type. It also points out where model definitions are inherited from previously defined models. The rationale of these calculations is found in the original *Conveyance System Cost Estimates Task 250 Report*.

Where formulas are displayed; items appearing within brackets with the `fixed` type-face denote *Variables* such as `<outer-diameter>`; items appearing within braces with the fixed `type-face` font denote *Domain Parameters* input by the user such as `{length}`; items appearing within braces in **`bold fixed`** denote *Domain Constants* such as **`{spoils-haul-unitcost}`**; items appearing in **Bold** or in **"bold quotes"** are literals such as numbers and constants, and items appearing in **`[Bracked Bold fixed type-face]`** are references to lookup tables.

Horizontally Drilled Pipe

Programatic name: **`hdtunnel`**
Specialized User Interface: `tabula.model.trenchless.HDTunnelInfoFrame`
Customized Icon Path: `tabula/model/tunnel/hdtunnel_icon.gif`

Domain Definition

Description: *Horizontally Drilled Pipe*

Summary

Constant List	
Name	**Defined In**
`spoil-haul-unitcost`	`hdtunnel`

Parameter List		
Name	**Description**	**Defined In**
`casing-required`	Casing Required	`hdtunnel`
`constrained-pipe-space`	Constrained Pipe Side Space	`hdtunnel`
`constrained-rig-space`	Constrained Rig Side Space	`hdtunnel`

construction-year	Construction Year	hdtunnel
diameter	Pipe Diameter	hdtunnel
easment-length	Total Easement Length	hdtunnel
easment-type	Easement Type	hdtunnel
gravels-and-cobbles	Gravels and Cobbles Present	hdtunnel
length	Length	hdtunnel
ps-surface-restoration	Pipe Side Surface Restoration	hdtunnel
ps-utilities	Pipe Side Existing Utilities	hdtunnel
rs-surface-restoration	Rig Side Surface Restoration	hdtunnel
rs-utilities	Rig Side Existing Utilities	hdtunnel

Cross Reference Lookup Table List	
Name	**Defined In**
casing-drill-unitcost	hdtunnel
casing-inner-diameter	hdtunnel
casing-material-unitcost	hdtunnel
casing-outer-diameter	hdtunnel
normal-drill-unitcost	hdtunnel
normal-outer-diameter	hdtunnel
property-row-unitcost	hdtunnel
surface-restoration-unitcost	hdtunnel
tunneling-row-unitcost	hdtunnel
utilities-unitcost	hdtunnel

Constant Definitions

Name	Type	Value	Units
`spoil-haul-unitcost`	cost	25	$/CY

Parameter Definitions

Name	Description	Type	Default	Units
`casing-required`	Casing Required	boolean	false	n/a
`constrained-pipe-space`	Constrained Pipe Side Space	boolean	false	n/a
`constrained-rig-space`	Constrained Rig Side Space	boolean	false	n/a
`construction-year`	Construction Year	year	2006	n/a
`diameter`	Pipe Diameter	enum	12	n/a
		Valid Values:	6 in. = 6 12 in. = 12 15 in. = 15 18 in. = 18 21 in. = 21 24 in. = 24 30 in. = 30 36 in. = 36 42 in. = 42 48 in. = 48	
`easment-length`	Total Easement Length	length	0	ft
`easment-type`	Easement Type	enum	none	n/a
		Valid Values:	None = none Residential = residential Industrial = industrial Office/Commercial = commercial	
`gravels-and-cobbles`	Gravels and Cobbles Present	boolean	false	n/a

`length`	Length	length	100	ft
			(Valid values range to 8000.)	
`ps-surface-restoration`	Pipe Side Surface Restoration	enum	hydroseed	n/a
		Valid Values:	None = none	
			Hydroseed = hydroseed	
			Pavement = pavement	
`ps-utilities`	Pipe Side Existing Utilities	enum	average	n/a
		Valid Values:	None = none	
			Average = average	
			Complex = complex	
`rs-surface-restoration`	Rig Side Surface Restoration	enum	hydroseed	n/a
		Valid Values:	None = none	
			Hydroseed = hydroseed	
			Pavement = pavement	
`rs-utilities`	Rig Side Existing Utilities	enum	average	n/a
		Valid Values:	None = none	
			Average = average	
			Complex = complex	

Cross Reference Table Definitions

casing-drill-unitcost	
Value (in)	**Result** ($/ft)
6	180
12	270
15	375
18	470
21	530

24	640
30	750
36	890
42	1000
48	NaN

casing-inner-diameter	
Value (in)	**Result** (in)
6	12
12	15
15	18
18	21
21	24
24	30
30	36
36	42
42	48
48	NaN

casing-material-unitcost	
Value (in)	**Result** ($/lf)
6	19
12	31
15	40
18	47

21	58
24	91
30	109
36	127
42	146
48	NaN

casing-outer-diameter	
Value (in)	**Result** (ft)
6	1.06
12	1.33
15	1.58
18	1.83
21	2.08
24	2.58
30	3.16
36	3.66
42	4.16
48	NaN

normal-drill-unitcost	
Value (in)	**Result** ($/ft)
6	60
12	180
15	270

18	375
21	470
24	530
30	640
36	750
42	890
48	1000

normal-outer-diameter	
Value (in)	**Result** (ft)
6	0.54
12	1.06
15	1.33
18	1.58
21	1.83
24	2.08
30	2.58
36	3.16
42	3.66
48	4.16

property-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	58

industrial	23
commercial	37

surface-restoration-unitcost	
Value (enum)	**Result** ($/SY)
none	0
hydroseed	5
pavement	40

tunneling-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	17
industrial	7
commercial	11

utilities-unitcost	
Value (enum)	**Result** ($/SF)
none	0
average	7
complex	12

Calculator Definition

Description: *Horizontally Drilled Pipe 1.0*
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`casing-diameter`	Casing Diameter	`hdtunnel`
`outer-diameter`	Outer Diameter	`hdtunnel`
`pit-footprint`	Slurry Pit Footprint	`hdtunnel`
`pit-length`	Slurry Pit Length	`hdtunnel`
`pit-width`	Slurry Pit Width	`hdtunnel`
`tunnel-spoils`	Spoils Volume	`hdtunnel`

Line Item Cost List		
Name	**Description**	**Defined In**
`casing-pipe-cost`	Casing Pipe Material Cost	`hdtunnel`
`drilling-cost`	Drilling Cost	`hdtunnel`
`ps-easment-cost`	Pipe Side Easement	`hdtunnel`
`ps-surface-restoration-cost`	Pipe Side Surface Restoration	`hdtunnel`
`ps-utilites-cost`	Pipe Side Existing Utilities	`hdtunnel`
`rs-easment-cost`	Rig Side Easement	`hdtunnel`
`rs-surface-restoration-cost`	Rig Side Surface Restoration	`hdtunnel`
`rs-utilites-cost`	Rig Side Existing Utilities	`hdtunnel`
`spoils-haul-cost`	Spoils Haul	`hdtunnel`
`tunnel-easment-cost`	Microtunnel Easement	`hdtunnel`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`constrained-pipe-side-space`	Constrained Pipe Side Space (10 percent if required)	`hdtunnel`
`constrained-rig-side-space`	Constrained Rig Side Space (10 percent if required)	`hdtunnel`
`gravel-multiplier`	Gravel and Cobbles (20 percent if required)	`hdtunnel`
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`hdtunnel`

Variable Definitions

casing-diameter
Category: *Geometry* Description: *Casing Diameter* Units: `in`
Variable = (*if*[{`casing-required`}=**"false"**, **NaN**, **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-inner-diameter`]**), **0**])

outer-diameter
Category: *Geometry* Description: *Outer Diameter* Units: `ft`
Variable = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-outer-diameter`]**), **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-outer-diameter`]**), **0**])

pit-footprint
Category: *Geometry* Description: *Slurry Pit Footprint* Units: `SF`
Variable = (`<pit-width> X <pit-length>`)

pit-length
Category: *Geometry* Description: *Slurry Pit Length* Units: `ft`
Variable = *ceiling*([({`diameter`} `X` [**1**/ **12**]) `+` **19**])

pit-width
Category: *Geometry* Description: *Slurry Pit Width* Units: `ft`
Variable = *ceiling*([({`diameter`} `X` [**1**/ **12**]) `+` **13**])

tunnel-spoils
Category: *Geometry* Description: *Spoils Volume* Units: `CY`
Variable = (`<outer-diameter> X <outer-diameter> X` {`length`} `X` **3.141592654** `X` [**1**/ **108**])

Line Item Cost Definitions

casing-pipe-cost

Description: *Cased Carrier Pipe Cost* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = *if*({`casing-required`}=**"true"**, [*lookup* {`diameter`} in **[`casing-material-unitcost`]**], **0**)

drilling-cost
Description: *Drilling Cost* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-drill-unitcost`]**), **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-drill-unitcost`]**), **0**])

ps-easment-cost
Description: *Pipe Side Easement* Unit Value: `SF`
Quantity = `<pit-footprint>`
Unit Cost = (*lookup* {`easment-type`} in **[`property-row-unitcost`]**)

ps-surface-restoration-cost
Description: *Pipe Side Surface Restoration* Unit Value: `SY`
Quantity = (`<pit-footprint>` `X` [**1**/ **9**])
Unit Cost = (*lookup* {`ps-surface-restoration`} in **[`surface-restoration-unitcost`]**)

ps-utilites-cost
Description: *Pipe Side Existing Utilities* Unit Value: `SF`
Quantity = `<pit-footprint>`
Unit Cost = (*lookup* {`ps-utilities`} in **[`utilities-unitcost`]**)

rs-easment-cost
Description: *Rig Side Easement* Unit Value: `SF`
Quantity = `<pit-footprint>`
Unit Cost = (*lookup* {`easment-type`} in **[`property-row-unitcost`]**)

rs-surface-restoration-cost
Description: *Rig Side Surface Restoration* Unit Value: `SY`
Quantity = (`<pit-footprint> X` [**1**/ **9**])
Unit Cost = (*lookup* {`rs-surface-restoration`} in **[`surface-restoration-unitcost`]**)

rs-utilites-cost
Description: *Rig Side Existing Utilities* Unit Value: `SF`
Quantity = `<pit-footprint>`
Unit Cost = (*lookup* {`rs-utilities`} in **[`utilities-unitcost`]**)

spoils-haul-cost
Description: *Spoils Haul* Unit Value: `CY`
Quantity = `<tunnel-spoils>`
Unit Cost = **`{spoil-haul-unitcost}`**

tunnel-easment-cost
Description: *Microtunnel Easement* Unit Value: `SF`
Quantity = (`{easment-length}` `X` [*ceiling*([`{diameter}` `X` (**1**/ **12**)]) `+` **20**])
Unit Cost = (*lookup* `{easment-type}` in **`[tunneling-row-unitcost]`**)

Multiplier Definitions

constrained-pipe-side-space
Description: *Constrained Pipe Side Space (10 percent if required)*
Multiplier = (**1** `+` *if*[`{constrained-pipe-space}`=**"true"**, **0.1**, **0**])

constrained-rig-side-space
Description: *Constrained Rig Side Space (10 percent if required)*
Multiplier = (**1** `+` *if*[`{constrained-rig-space}`=**"true"**, **0.1**, **0**])

gravel-multiplier

Description: *Gravel and Cobbles (20 percent if required)*
Multiplier = (**1** + *if*[{`gravels-and-cobbles`}=**"true"**, **0.2**, **0**])

mobilization-demobilization
Description: *Mobilization/Demobilization at 10 percent*
Multiplier = **1.1**

Year Definition

Year Calculation
Year = {`construction-year`}

Jack and Bore

Programatic name: **`jbtunnel`**
Specialized User Interface: `tabula.model.trenchless.TunnelInfoFrame`
Customized Icon Path: `tabula/model/tunnel/jbtunnel_icon.gif`

Domain Definition

Description: *Jack and Bore*
Inherits from *Microtunnel Template* `(microtunnel-template)`

Summary

Constant List	
Name	**Defined In**
`backfill-unitcost`	`microtunnel-template`
`excavation-and-backfill-unitcost`	`jbtunnel`
`shaft-excavation-unitcost`	`microtunnel-template`
`spoil-haul-unitcost`	`microtunnel-template`

Parameter List		
Name	**Description**	**Defined In**
`casing-required`	Casing Required	`microtunnel-template`
`construction-year`	Construction Year	`trenchless-conduit-template`
`dewatering`	Dewatering	`jbtunnel`
`diameter`	Inside Diameter	`jbtunnel`
`easment-length`	Tunnel Easement Length	`trenchless-conduit-template`
`easment-type`	Easement Type	`trenchless-conduit-template`
`length`	Length	`jbtunnel`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-surface-restoration`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilities`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-surface-restoration`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilities`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`traffic`	Traffic	`microtunnel-template`

Cross Reference Lookup Table List	
Name	**Defined In**

casing-material-unitcost	microtunnel-template
casing-outer-diameter	microtunnel-template
casing-pipe-diameter	microtunnel-template
casing-variable-unitcost	jbtunnel
normal-outer-diameter	microtunnel-template
normal-variable-unitcost	jbtunnel
property-row-unitcost	microtunnel-template
surface-restoration-unitcost	trenchless-conduit-template
traffic-unitcost	microtunnel-template
tunneling-row-unitcost	microtunnel-template
utilities-unitcost	trenchless-conduit-template

Constant Definitions

Name	Type	Value	Units
excavation-and-backfill-unitcost	cost	33	$/CY

Constants inherited from **microtunnel-template**.
backfill-unitcost, shaft-excavation-unitcost, spoil-haul-unitcost

Parameter Definitions

Name	Description	Type	Default	Units
dewatering	Dewatering	enum	minimal	n/a
		Valid Values:	None = none Minimal = minimal	
diameter	Inside Diameter	enum	12	n/a

			Valid Values:	12 in. = 12 15 in. = 15 18 in. = 18 21 in. = 21 24 in. = 24 30 in. = 30 36 in. = 36 42 in. = 42 48 in. = 48 54 in. = 54 60 in. = 60 66 in. = 66 72 in. = 72 84 in. = 84 90 in. = 90 96 in. = 96 108 in. = 108 120 in. = 120
`length`	Length	length	100	ft
			(Valid values range to 300.)	

Parameters inherited from **`microtunnel-template`**.
`casing-required, traffic`

Parameters inherited from **`trenchless-conduit-template`**.
`construction-year, easment-length, easment-type, ls-excavation-depth, ls-surface-restoration, ls-utilities, rs-excavation-depth, rs-surface-restoration, rs-utilities`

Cross Reference Table Definitions

casing-variable-unitcost	
Value (in)	**Result** ($/lf)
12	408
15	540
18	540
21	720
24	720
30	924
36	1056
42	1188
48	1380
54	1944
60	1944
66	2352
72	2352
84	2784
90	3132
96	3132
108	3600
120	4320

normal-variable-unitcost	
Value (in)	**Result** ($/lf)
12	240
15	270

18	306
21	357
24	408
30	540
36	720
42	924
48	1056
54	1188
60	1380
66	1518
72	1944
84	2352
90	2520
96	2784
108	3132
120	3600

Cross Reference Tables inherited from **`microtunnel-template`**.
`casing-material-unitcost, casing-outer-diameter, casing-pipe-diameter, normal-outer-diameter, property-row-unitcost, traffic-unitcost, tunneling-row-unitcost`

Cross Reference Tables inherited from **`trenchless-conduit-template`**.
`surface-restoration-unitcost, utilities-unitcost`

Calculator Definition

Description: *Jack and Bore 1.0*
Inherits from *Microtunnel Template 1.0* `(microtunnel-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`casing-pipe-diameter`	Casing Pipe Diameter	`jbtunnel`
`ls-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`ls-easment-length`	Launch Shaft Easement Length	`trenchless-conduit-template`
`ls-easment-width`	Launch Shaft Easement Width	`trenchless-conduit-template`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-footprint`	Footprint	`trenchless-conduit-template`
`ls-length`	Length	`jbtunnel`
`ls-volume`	Volume	`trenchless-conduit-template`
`ls-width`	Width	`jbtunnel`
`outer-diameter`	Outer Diameter	`jbtunnel`
`rs-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`rs-easment-length`	Retrieval Shaft Easement Length	`trenchless-conduit-template`
`rs-easment-width`	Retrieval Shaft Easement Width	`trenchless-conduit-template`
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`

`rs-footprint`	Footprint	`trenchless-conduit-template`
`rs-length`	Length	`jbtunnel`
`rs-volume`	Volume	`trenchless-conduit-template`
`rs-width`	Width	`jbtunnel`
`tunnel-spoil-loads`	Spoils Loads	`trenchless-conduit-template`
`tunnel-spoils`	Spoils Volume	`trenchless-conduit-template`

Line Item Cost List		
Name	**Description**	**Defined In**
`casing-pipe-cost`	Casing Pipe Material Cost	`microtunnel-template`
`dewatering-cost`	Tunnel Dewatering	`jbtunnel`
`ls-backfill-cost`	(not used)	`jbtunnel`
`ls-easment-cost`	Launch Shaft Easement	`trenchless-conduit-template`
`ls-excavation-cost`	Launch Shaft Excavation/Backfill	`jbtunnel`
`ls-shoring-cost`	Launch Shaft Shoring	`trenchless-conduit-template`
`ls-surface-restoration-cost`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilites-cost`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-backfill-cost`	(not used)	`jbtunnel`
`rs-easment-cost`	Retrieval Shaft Easement	`trenchless-conduit-template`
`rs-excavation-cost`	Retrieval Shaft Excavation/Backfill	`jbtunnel`

`rs-shoring-cost`	Retrieval Shaft Shoring	`trenchless-conduit-template`
`rs-surface-restoration-cost`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilites-cost`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`spoils-haul-cost`	Spoils Haul	`trenchless-conduit-template`
`traffic-cost`	Traffic Control	`jbtunnel`
`tunnel-boring-cost`	Boring Cost	`jbtunnel`
`tunnel-easment-cost`	Microtunnel Easement	`microtunnel-template`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenchless-conduit-template`

Variable Definitions

casing-pipe-diameter
Category: *Tunnel Geometry* Description: *Casing Pipe Diameter* Units: `in`
Variable = (*if*[{`casing-required`}=**"false"**, **NaN**, **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-pipe-diameter`]**), **0**])

ls-length
Category: *Launch Shaft Geometry* Description: *Length*

Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **24**)

ls-width
Category: *Launch Shaft Geometry* Description: *Width* Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **13**)

outer-diameter
Category: *Tunnel Geometry* Description: *Outer Diameter* Units: `ft`
Variable = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-outer-diameter`]**), **0**] `+` *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-outer-diameter`]**), **0**])

rs-length
Category: *Retrieval Shaft Geometry* Description: *Length* Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **19**)

rs-width
Category: *Retrieval Shaft Geometry* Description: *Width* Units: `ft`

Variable = (*ceiling*[({diameter} X [**1**/ **12**])] + **13**)

Variables inherited from **trenchless-conduit-template**.
ls-easment-footprint, ls-easment-length, ls-easment-width, ls-excavation-depth, ls-footprint, ls-volume, rs-easment-footprint, rs-easment-length, rs-easment-width, rs-excavation-depth, rs-footprint, rs-volume, tunnel-spoil-loads, tunnel-spoils

Line Item Cost Definitions

dewatering-cost
Description: *Tunnel Dewatering* Unit Value: LS
Quantity = **1**
Unit Cost = (*if*[{dewatering}=**"minimal"**, **8500**, **0**])

ls-backfill-cost
Description: *(not used)* Unit Value: CY
Quantity = **0**
Unit Cost = **0**

ls-excavation-cost
Description: *Launch Shaft Excavation/Backfill* Unit Value: CY
Quantity = <ls-volume>
Unit Cost = **{excavation-and-backfill-unitcost}**

rs-backfill-cost
Description: *(not used)* Unit Value: `CY`
Quantity = **0**
Unit Cost = **0**

rs-excavation-cost
Description: *Retrieval Shaft Excavation/Backfill* Unit Value: `CY`
Quantity = `<rs-volume>`
Unit Cost = **`{excavation-and-backfill-unitcost}`**

traffic-cost
Description: *Traffic Control* Unit Value: `shaft`
Quantity = **2**
Unit Cost = (*lookup* {`traffic`} in **[`traffic-unitcost`]**)

tunnel-boring-cost
Description: *Boring Cost* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-variable-unitcost`]**), **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-variable-unitcost`]**), **0**])

Line item costs inherited from **`microtunnel-template`**.
`casing-pipe-cost, tunnel-easment-cost`

Line item costs inherited from **`trenchless-conduit-template`**.
`ls-easment-cost, ls-shoring-cost, ls-surface-restoration-cost, ls-utilites-cost, rs-easment-cost, rs-shoring-cost, rs-surface-restoration-cost, rs-utilites-cost, spoils-haul-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenchless-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = {`construction-year`}

Microtunnel

Programatic name: **`mtunnel`**
Specialized User Interface: `tabula.model.trenchless.MTunnelInfoFrame`
Customized Icon Path: `tabula/model/tunnel/mtunnel_icon.gif`

Domain Definition

Description: *Microtunnel*
Inherits from *Microtunnel Template* `(microtunnel-template)`

Summary

Constant List	
Name	**Defined In**
`backfill-unitcost`	`microtunnel-template`
`shaft-excavation-unitcost`	`microtunnel-template`
`spoil-haul-unitcost`	`microtunnel-template`

Parameter List		
Name	**Description**	**Defined In**
`casing-required`	Casing Required	`microtunnel-template`
`construction-year`	Construction Year	`trenchless-conduit-template`
`dewatering`	Dewatering	`trenchless-conduit-template`
`diameter`	Inside Diameter	`mtunnel`
`easment-length`	Tunnel Easement Length	`trenchless-conduit-template`
`easment-type`	Easement Type	`trenchless-conduit-template`
`i-excavation-depth`	Intermediate Shaft Excavation Depth	`mtunnel`
`i-number`	Number of Intermediate Shafts	`mtunnel`
`i-surface-restoration`	Intermediate Shaft Surface Restoration	`mtunnel`
`i-utilities`	Intermediate Shaft Existing Utilities	`mtunnel`
`length`	Length	`trenchless-conduit-template`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`

`ls-surface-restoration`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilities`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-surface-restoration`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilities`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`traffic`	Traffic	`microtunnel-template`

Cross Reference Lookup Table List	
Name	**Defined In**
`casing-fixed-unitcost`	`Mtunnel`
`casing-material-unitcost`	`microtunnel-template`
`casing-outer-diameter`	`microtunnel-template`
`casing-pipe-diameter`	`microtunnel-template`
`casing-variable-unitcost`	`Mtunnel`
`normal-fixed-unitcost`	`Mtunnel`
`normal-outer-diameter`	`microtunnel-template`
`normal-variable-unitcost`	`Mtunnel`
`property-row-unitcost`	`microtunnel-template`
`surface-restoration-unitcost`	`trenchless-conduit-template`
`traffic-unitcost`	`microtunnel-template`
`tunneling-row-unitcost`	`microtunnel-template`
`utilities-unitcost`	`trenchless-conduit-template`

Constant Definitions

No Defined Constants

Constants inherited from **`microtunnel-template`**.
`backfill-unitcost, shaft-excavation-unitcost, spoil-haul-unitcost`

Parameter Definitions

Name	Description	Type	Default	Units
`diameter`	Inside Diameter	enum	12	n/a
		Valid Values:	12 in. = 12 15 in. = 15 18 in. = 18 21 in. = 21 24 in. = 24 30 in. = 30 36 in. = 36 42 in. = 42 48 in. = 48 54 in. = 54 60 in. = 60 66 in. = 66 72 in. = 72 84 in. = 84	
`i-excavation-depth`	Intermediate Shaft Excavation Depth	length	40	ft
		(Valid values range from 15.)		
`i-number`	Number of Intermediate Shafts	decimal	0	n/a
		(Valid values range from 0.)		
`i-surface-restoration`	Intermediate Shaft Surface	enum	hydroseed	n/a

	Restoration			
			Valid Values:	None = none Hydroseed = hydroseed Pavement = pavement
`i-utilities`	Intermediate Shaft Existing Utilities	enum	average	n/a
			Valid Values:	None = none Average = average Complex = complex

Parameters inherited from **`microtunnel-template`**.
`casing-required, traffic`

Parameters inherited from **`trenchless-conduit-template`**.
`construction-year, dewatering, easment-length, easment-type, length, ls-excavation-depth, ls-surface-restoration, ls-utilities, rs-excavation-depth, rs-surface-restoration, rs-utilities`

Cross Reference Table Definitions

casing-fixed-unitcost	
Value (in)	**Result** ($)
12	210000
15	270000
18	270000
21	330000
24	330000
30	400000

36	470000
42	540000
48	600000
54	670000
60	740000
66	800000
72	800000
84	NaN

casing-variable-unitcost	
Value (in)	**Result** ($/lf)
12	744
15	900
18	900
21	1044
24	1044
30	1176
36	1296
42	1458
48	1560
54	1800
60	1800
66	2016
72	2016
84	NaN

normal-fixed-unitcost	
Value (in)	**Result** ($)
12	120000
15	130000
18	160000
21	190000
24	210000
30	270000
36	330000
42	400000
48	470000
54	540000
60	600000
66	670000
72	740000
84	800000

normal-variable-unitcost	
Value (in)	**Result** ($/lf)
12	480
15	555
18	630
21	672
24	744
30	900
36	1044

42	1176
48	1296
54	1458
60	1560
66	1650
72	1800
84	2016

Cross Reference Tables inherited from **`microtunnel-template`**.
`casing-material-unitcost, casing-outer-diameter, casing-pipe-diameter, normal-outer-diameter, property-row-unitcost, traffic-unitcost, tunneling-row-unitcost`

Cross Reference Tables inherited from **`trenchless-conduit-template`**.
`surface-restoration-unitcost, utilities-unitcost`

Calculator Definition

Description: *Microtunnel 1.0*
Inherits from *Microtunnel Template 1.0* `(microtunnel-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`casing-pipe-diameter`	Casing Pipe Diameter	`mtunnel`
`i-easment-footprint`	Easement Footprint	`mtunnel`

`i-easment-length`	Intermedate Shaft Easement Length	`mtunnel`
`i-easment-width`	Intermediate Shaft Easement Width	`mtunnel`
`i-footprint`	Footprint	`mtunnel`
`i-length`	Length	`mtunnel`
`i-volume`	Volume	`mtunnel`
`i-width`	Width	`mtunnel`
`ls-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`ls-easment-length`	Launch Shaft Easement Length	`trenchless-conduit-template`
`ls-easment-width`	Launch Shaft Easement Width	`trenchless-conduit-template`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-footprint`	Footprint	`trenchless-conduit-template`
`ls-length`	Length	`mtunnel`
`ls-volume`	Volume	`trenchless-conduit-template`
`ls-width`	Width	`mtunnel`
`outer-diameter`	Outer Diameter	`mtunnel`
`rs-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`rs-easment-length`	Retrieval Shaft Easement Length	`trenchless-conduit-template`
`rs-easment-width`	Retrieval Shaft Easement Width	`trenchless-conduit-template`
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-footprint`	Footprint	`trenchless-conduit-template`

`rs-length`	Length	`mtunnel`
`rs-volume`	Volume	`trenchless-conduit-template`
`rs-width`	Width	`mtunnel`
`total-shafts`	Total Number of Shafts	`mtunnel`
`tunnel-spoil-loads`	Spoils Loads	`trenchless-conduit-template`
`tunnel-spoils`	Spoils Volume	`trenchless-conduit-template`

Line Item Cost List		
Name	**Description**	**Defined In**
`casing-pipe-cost`	Casing Pipe Material Cost	`microtunnel-template`
`dewatering-cost`	Tunnel Dewatering	`mtunnel`
`i-backfill-cost`	Intermediate Shaft Backfill	`mtunnel`
`i-easment-cost`	Intermediate Shaft Easement	`mtunnel`
`i-excavation-cost`	Intermediate Shaft Excavation	`mtunnel`
`i-shoring-cost`	Intermediate Shaft Shoring	`mtunnel`
`i-surface-restoration-cost`	Intermediate Shaft Surface Restoration	`mtunnel`
`i-utilites-cost`	Intermediate Shaft Existing Utilities	`mtunnel`
`ls-backfill-cost`	Launch Shaft Backfill	`trenchless-conduit-template`
`ls-easment-cost`	Launch Shaft Easement	`trenchless-conduit-template`
`ls-excavation-cost`	Launch Shaft Excavation	`trenchless-conduit-template`
`ls-shoring-cost`	Launch Shaft Shoring	`trenchless-conduit-template`

`ls-surface-restoration-cost`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilites-cost`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`mtmb-procurment-cost`	MTBM Fixed Costs	`mtunnel`
`rs-backfill-cost`	Retrieval Shaft Backfill	`trenchless-conduit-template`
`rs-easment-cost`	Retrieval Shaft Easement	`trenchless-conduit-template`
`rs-excavation-cost`	Retrieval Shaft Excavation	`trenchless-conduit-template`
`rs-shoring-cost`	Retrieval Shaft Shoring	`trenchless-conduit-template`
`rs-surface-restoration-cost`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilites-cost`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`spoils-haul-cost`	Spoils Haul	`trenchless-conduit-template`
`traffic-cost`	Traffic Control	`mtunnel`
`tunnel-boring-cost`	Microtunnel Boring	`mtunnel`
`tunnel-easment-cost`	Microtunnel Easement	`microtunnel-template`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenchless-conduit-template`

Variable Definitions

casing-pipe-diameter

Category: *Tunnel Geometry*
Description: *Casing Pipe Diameter*
Units: `in`

Variable = (*if*[{`casing-required`}=**"false"**, **NaN**, **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[casing-pipe-diameter]**), **0**])

i-easment-footprint

Category: *Intermediate Shaft Geometry*
Description: *Easement Footprint*
Units: `SF`

Variable = (`<ls-easment-width> X <ls-easment-length>`)

i-easment-length

Description: *Intermedate Shaft Easement Length*
Units: `ft`

Variable = (**20** + *if*[`<ls-excavation-depth>` > **12**, **10**, **0**] + *if*[`<ls-excavation-depth>` > **20**, **20**, **0**] + `<ls-length>`)

i-easment-width

Description: *Intermediate Shaft Easement Width*
Units: `ft`

Variable = (**20** + *if*[`<ls-excavation-depth>` > **12**, **10**, **0**] + *if*[`<ls-excavation-depth>` > **20**, **20**, **0**] + `<ls-width>`)

i-footprint

Category: *Intermediate Shaft Geometry*
Description: *Footprint*

Units: `SF`
Variable = (`<ls-width> X <ls-length>`)

i-length
Category: *Intermediate Shaft Geometry* Description: *Length* Units: `ft`
Variable = `<ls-length>`

i-volume
Category: *Intermediate Shaft Geometry* Description: *Volume* Units: `CY`
Variable = (`<ls-footprint> X {ls-excavation-depth} X` [**1**/ **27**])

i-width
Category: *Intermediate Shaft Geometry* Description: *Width* Units: `ft`
Variable = `<ls-width>`

ls-length
Category: *Launch Shaft Geometry* Description: *Length* Units: `ft`
Variable = (*ceiling*[(`{diameter} X` [**1**/ **12**])] `+` **28**)

ls-width
Category: *Launch Shaft Geometry* Description: *Width* Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **15**)

outer-diameter
Category: *Tunnel Geometry* Description: *Outer Diameter* Units: `ft`
Variable = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-outer-diameter`]**), **0**] `+` *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-outer-diameter`]**), **0**])

rs-length
Category: *Retrieval Shaft Geometry* Description: *Length* Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **19**)

rs-width
Category: *Retrieval Shaft Geometry* Description: *Width* Units: `ft`
Variable = (*ceiling*[({`diameter`} `X` [**1**/ **12**])] `+` **19**)

total-shafts
Description: *Total Number of Shafts*
Variable = (**2** + {i-number})

Variables inherited from **trenchless-conduit-template**.
ls-easment-footprint, ls-easment-length, ls-easment-width, ls-excavation-depth, ls-footprint, ls-volume, rs-easment-footprint, rs-easment-length, rs-easment-width, rs-excavation-depth, rs-footprint, rs-volume, tunnel-spoil-loads, tunnel-spoils

Line Item Cost Definitions

dewatering-cost
Description: *Tunnel Dewatering* Unit Value: LS
Quantity = **1**
Unit Cost = (*if*[{dewatering}=**"minimal"**, (**50000** + *if*[<total-shafts> > **2**, **5000**, **0**] + *if*[<total-shafts> > **3**, **5000**, **0**] + *if*[<total-shafts> > **4**, **10000**, **0**] + *if*[<total-shafts> > **5**, **20000**, **0**]), **0**] + *if*[{dewatering}=**"significant"**, (**70000** + *if*[<total-shafts> > **2**, **10000**, **0**] + *if*[<total-shafts> > **3**, **25000**, **0**] + *if*[<total-shafts> > **4**, **15000**, **0**] + *if*[<total-shafts> > **5**, **20000**, **0**]), **0**])

i-backfill-cost
Description: *Intermediate Shaft Backfill* Unit Value: CY
Quantity = ({i-number} X <i-volume>)
Unit Cost = **{backfill-unitcost}**

i-easment-cost
Description: *Intermediate Shaft Easement* Unit Value: `SF`
Quantity = ({`i-number`} `X` `<i-footprint>`)
Unit Cost = (*lookup* {`easment-type`} in **[`property-row-unitcost`]**)

i-excavation-cost
Description: *Intermediate Shaft Excavation* Unit Value: `CY`
Quantity = ({`i-number`} `X` `<i-volume>`)
Unit Cost = **{`shaft-excavation-unitcost`}**

i-shoring-cost
Description: *Intermediate Shaft Shoring* Unit Value: `SF`
Quantity = ({`i-number`} `X` **2** `X` {`i-excavation-depth`} `X` [`<i-width>` `+` `<i-length>`])
Unit Cost = (**10** `+` [**1.85** `X` {`i-excavation-depth`}])

i-surface-restoration-cost
Description: *Intermediate Shaft Surface Restoration* Unit Value: `SY`
Quantity = ({`i-number`} `X` `<i-footprint>` `X` [**1**/ **9**])
Unit Cost = (*lookup* {`i-surface-restoration`} in **[`surface-restoration-unitcost`]**)

i-utilites-cost
Description: *Intermediate Shaft Existing Utilities* Unit Value: `SF`
Quantity = ({`i-number`} `X` `<i-footprint>`)
Unit Cost = (*lookup* {`i-utilities`} in **[`utilities-unitcost`]**)

mtmb-procurment-cost
Description: *MTBM Fixed Costs* Unit Value: `LS`
Quantity = **1**
Unit Cost = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-fixed-unitcost`]**), **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-fixed-unitcost`]**), **0**])

traffic-cost
Description: *Traffic Control* Unit Value: `shaft`
Quantity = `<total-shafts>`
Unit Cost = (*lookup* {`traffic`} in **[`traffic-unitcost`]**)

tunnel-boring-cost
Description: *Microtunnel Boring* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = (*if*[{`casing-required`}=**"false"**, (*lookup* {`diameter`} in **[`normal-variable-unitcost`]**), **0**] + *if*[{`casing-required`}=**"true"**, (*lookup* {`diameter`} in **[`casing-variable-unitcost`]**), **0**])

Line item costs inherited from **`microtunnel-template`**.
`casing-pipe-cost, tunnel-easment-cost`

Line item costs inherited from **`trenchless-conduit-template`**.
`ls-backfill-cost, ls-easment-cost, ls-excavation-cost, ls-shoring-cost, ls-surface-restoration-cost, ls-utilites-cost, rs-backfill-cost, rs-easment-cost, rs-excavation-cost, rs-shoring-cost, rs-surface-restoration-cost, rs-utilites-cost, spoils-haul-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenchless-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = `{construction-year}`

Microtunnel Template

Programatic name: **`microtunnel-template`**
Not Public, hidden from user interface

Domain Definition

Description: *Microtunnel Template*
Inherits from *Trenchless Conduit Template* `(trenchless-conduit-template)`

Summary

Constant List	
Name	**Defined In**
backfill-unitcost	microtunnel-template
shaft-excavation-unitcost	microtunnel-template
spoil-haul-unitcost	microtunnel-template

Parameter List		
Name	**Description**	**Defined In**
casing-required	Casing Required	microtunnel-template
construction-year	Construction Year	trenchless-conduit-template
dewatering	Dewatering	trenchless-conduit-template
diameter	Diameter	trenchless-conduit-template
easment-length	Tunnel Easement Length	trenchless-conduit-template
easment-type	Easement Type	trenchless-conduit-template
length	Length	trenchless-conduit-template
ls-excavation-depth	Launch Shaft Excavation Depth	trenchless-conduit-template
ls-surface-restoration	Launch Shaft Surface Restoration	trenchless-conduit-template
ls-utilities	Launch Shaft Existing Utilities	trenchless-conduit-template
rs-excavation-depth	Retrieval Shaft Excavation Depth	trenchless-conduit-template
rs-surface-restoration	Retrieval Shaft Surface	trenchless-conduit-template

	Restoration	
`rs-utilities`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`traffic`	Traffic	`microtunnel-template`

Cross Reference Lookup Table List	
Name	**Defined In**
`casing-material-unitcost`	`microtunnel-template`
`casing-outer-diameter`	`microtunnel-template`
`casing-pipe-diameter`	`microtunnel-template`
`normal-outer-diameter`	`microtunnel-template`
`property-row-unitcost`	`microtunnel-template`
`surface-restoration-unitcost`	`trenchless-conduit-template`
`traffic-unitcost`	`microtunnel-template`
`tunneling-row-unitcost`	`microtunnel-template`
`utilities-unitcost`	`trenchless-conduit-template`

Constant Definitions

Name	Type	Value	Units
`backfill-unitcost`	cost	18	$/CY
`shaft-excavation-unitcost`	cost	15	$/CY
`spoil-haul-unitcost`	cost	12	$/CY

Parameter Definitions

Name	Description	Type	Default	Units

`casing-required`	Casing Required	boolean	false	n/a
`traffic`	Traffic	enum	heavy	n/a
		Valid Values:	None = none Light = standard Heavy = heavy	

Parameters inherited from **`trenchless-conduit-template`**.
`construction-year, dewatering, diameter, easment-length, easment-type, length, ls-excavation-depth, ls-surface-restoration, ls-utilities, rs-excavation-depth, rs-surface-restoration, rs-utilities`

Cross Reference Table Definitions

casing-material-unitcost	
Value (in)	**Result** ($/ft)
12	50
15	65
18	70
21	90
24	95
30	120
36	145
42	220
48	255
54	285
60	320
66	520
72	570

84	665
90	715
96	760
108	860
120	1270

casing-outer-diameter	
Value (in)	**Result** (ft)
12	2.5
15	3.08
18	3.08
21	3.66
24	3.66
30	4.25
36	4.83
42	5.54
48	6.12
54	7.29
60	7.29
66	8.33
72	8.33
84	9.62
90	10.66
96	10.66
108	11.66
120	14

casing-pipe-diameter	
Value (in)	**Result** (in)
12	24
15	30
18	30
21	36
24	36
30	42
36	48
42	54
48	60
54	72
60	72
66	84
72	84
84	96
90	108
96	108
108	120
120	144

normal-outer-diameter	
Value (in)	**Result** (ft)
12	1.41

15	1.66
18	1.91
21	2.2
24	2.5
30	3.08
36	3.66
42	4.25
48	4.83
54	5.54
60	6.12
66	7.29
72	7.29
84	8.33
90	9.62
96	9.62
108	10.66
120	11.66

property-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	58
industrial	23
commercial	37

traffic-unitcost	
Value (enum)	**Result** ($/shaft)
none	0
standard	20000
heavy	35000

tunneling-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	17
industrial	7
commercial	11

Cross Reference Tables inherited from **trenchless-conduit-template**.
surface-restoration-unitcost, utilities-unitcost

Calculator Definition

Description: *Microtunnel Template 1.0*
Inherits from *Trenched Conduit Template 1.0* (trenchless-conduit-template)
Basis Year: **2005**

Summary

Variable List		
Name	**Description**	**Defined In**
ls-easment-	Easement Footprint	trenchless-conduit-

footprint		template
ls-easment-length	Launch Shaft Easement Length	trenchless-conduit-template
ls-easment-width	Launch Shaft Easement Width	trenchless-conduit-template
ls-excavation-depth	Launch Shaft Excavation Depth	trenchless-conduit-template
ls-footprint	Footprint	trenchless-conduit-template
ls-length	LS Length Placeholder	trenchless-conduit-template
ls-volume	Volume	trenchless-conduit-template
ls-width	LS Width Placeholder	trenchless-conduit-template
outer-diameter	Outer Diameter Placeholder	trenchless-conduit-template
rs-easment-footprint	Easement Footprint	trenchless-conduit-template
rs-easment-length	Retrieval Shaft Easement Length	trenchless-conduit-template
rs-easment-width	Retrieval Shaft Easement Width	trenchless-conduit-template
rs-excavation-depth	Retrieval Shaft Excavation Depth	trenchless-conduit-template
rs-footprint	Footprint	trenchless-conduit-template
rs-length	RS Length Placeholder	trenchless-conduit-template
rs-volume	Volume	trenchless-conduit-template
rs-width	RS Width Placeholder	trenchless-conduit-template
tunnel-spoil-loads	Spoils Loads	trenchless-conduit-template
tunnel-spoils	Spoils Volume	trenchless-conduit-template

Line Item Cost List		
Name	**Description**	**Defined In**
`casing-pipe-cost`	Casing Pipe Material Cost	`microtunnel-template`
`ls-backfill-cost`	Launch Shaft Backfill	`trenchless-conduit-template`
`ls-easment-cost`	Launch Shaft Easement	`trenchless-conduit-template`
`ls-excavation-cost`	Launch Shaft Excavation	`trenchless-conduit-template`
`ls-shoring-cost`	Launch Shaft Shoring	`trenchless-conduit-template`
`ls-surface-restoration-cost`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilites-cost`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-backfill-cost`	Retrieval Shaft Backfill	`trenchless-conduit-template`
`rs-easment-cost`	Retrieval Shaft Easement	`trenchless-conduit-template`
`rs-excavation-cost`	Retrieval Shaft Excavation	`trenchless-conduit-template`
`rs-shoring-cost`	Retrieval Shaft Shoring	`trenchless-conduit-template`
`rs-surface-restoration-cost`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilites-cost`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`spoils-haul-cost`	Spoils Haul	`trenchless-conduit-template`
`tunnel-easment-cost`	Microtunnel Easement	`microtunnel-template`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenchless-conduit-template`

Variable Definitions

No Defined Variables

Variables inherited from **`trenchless-conduit-template`**.
`ls-easment-footprint, ls-easment-length, ls-easment-width, ls-excavation-depth, ls-footprint, ls-length, ls-volume, ls-width, outer-diameter, rs-easment-footprint, rs-easment-length, rs-easment-width, rs-excavation-depth, rs-footprint, rs-length, rs-volume, rs-width, tunnel-spoil-loads, tunnel-spoils`

Line Item Cost Definitions

casing-pipe-cost
Description: *Cased Carrier Pipe Cost* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = *if*({`casing-required`}=**"true"**, [*lookup* {`diameter`} in **[`casing-material-unitcost`]**], **0**)

tunnel-easment-cost
Description: *Microtunnel Easement* Unit Value: `SF`
Quantity = ({`easment-length`} `X` [*ceiling*([{`diameter`} `X` (**1**/ **12**)]) `+` **20**])
Unit Cost = (*lookup* {`easment-type`} in **[`tunneling-row-unitcost`]**)

Line item costs inherited from **`trenchless-conduit-template`**.
`ls-backfill-cost, ls-easment-cost, ls-excavation-cost, ls-shoring-cost, ls-surface-restoration-cost, ls-utilites-cost, rs-backfill-cost, rs-easment-cost, rs-excavation-cost, rs-shoring-cost, rs-surface-restoration-cost, rs-utilites-cost, spoils-haul-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenchless-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = {`construction-year`}

Parallel Pipes

Programatic name: **`ppipe`**
Specialized User Interface: `tabula.model.pipe.PipeInfoFrame`
Customized Icon Path: `tabula/model/pipe/tree_icon2.gif`

Domain Definition

Description: *Parallel Pipes*
Inherits from *Trenched Conduit Template* `(trenched-conduit-template)`

Summary

Constant List	
Name	**Defined In**

asphalt-paving-unitcost	trenched-conduit-template
excavation-unitcost	trenched-conduit-template
import-fill-unitcost	trenched-conduit-template
native-fill-unitcost	trenched-conduit-template
overlay-paving-unitcost	trenched-conduit-template
special-shoring-unitcost	trenched-conduit-template
spoil-load-and-haul-unitcost	trenched-conduit-template
trench-safety-unitcost	trenched-conduit-template

Parameter List

Name	Description	Defined In
backfill	Trench Backfill Type	trenched-conduit-template
construction-year	Construction Year	trenched-conduit-template
cover-depth	Depth of Cover	trenched-conduit-template
dewatering	Dewatering	trenched-conduit-template
diameter1	Pipe One Diameter	ppipe
diameter2	Pipe Two Diameter	ppipe
easment	Required Easements	trenched-conduit-template
existing-utilities	Existing Utilities	trenched-conduit-template
length	Length	trenched-conduit-template
manhole-spacing	Manhole Spacing	trenched-conduit-template
pavement-restoration	Pavement Restoration	trenched-conduit-template
row	Land Acquisition	trenched-conduit-template
traffic	Traffic	trenched-conduit-template
trench-safety	Trench Safety	trenched-conduit-template

`type`	Conduit Type	`trenched-conduit-template`

Cross Reference Lookup Table List	
Name	**Defined In**
`average-traffic-unitcost`	`trenched-conduit-template`
`average-utility-unitcost`	`trenched-conduit-template`
`complex-utility-unitcost`	`trenched-conduit-template`
`concrete-material`	`trenched-conduit-template`
`concrete-odiam`	`trenched-conduit-template`
`dewatering-well-unitcost`	`trenched-conduit-template`
`force-main-material`	`trenched-conduit-template`
`force-main-odiam`	`trenched-conduit-template`
`heavy-traffic-unitcost`	`trenched-conduit-template`
`high-force-main-material`	`trenched-conduit-template`
`installation-cost`	`trenched-conduit-template`
`manhole-additional-cost`	`trenched-conduit-template`
`manhole-base-cost`	`trenched-conduit-template`
`pavement-restoration-width`	`trenched-conduit-template`
`row-unitcost`	`trenched-conduit-template`
`trench-sump-dewatering-unitcost`	`trenched-conduit-template`

Constant Definitions

No Defined Constants

Constants inherited from **`trenched-conduit-template`**.
`asphalt-paving-unitcost, excavation-unitcost, import-fill-unitcost, native-fill-unitcost, overlay-paving-unitcost, special-shoring-`

```
unitcost, spoil-load-and-haul-unitcost, trench-safety-unitcost
```

Parameter Definitions

Name	Description	Type	Default	Units
diameter1	Pipe One Diameter	enum	12	n/a
			Valid Values:	8 in. = 8
				10 in. = 10
				12 in. = 12
				14 in. = 14
				15 in. = 15
				16 in. = 16
				18 in. = 18
				20 in. = 20
				21 in. = 21
				24 in. = 24
				27 in. = 27
				30 in. = 30
				36 in. = 36
				42 in. = 42
				48 in. = 48
				54 in. = 54
				60 in. = 60
				72 in. = 72
				78 in. = 78
				84 in. = 84
				96 in. = 96
				108 in. = 108
				120 in. = 120
				144 in. = 144
diameter2	Pipe Two Diameter	enum	12	n/a

Valid Values:	8 in. = 8
	10 in. = 10
	12 in. = 12
	14 in. = 14
	15 in. = 15
	16 in. = 16
	18 in. = 18
	20 in. = 20
	21 in. = 21
	24 in. = 24
	27 in. = 27
	30 in. = 30
	36 in. = 36
	42 in. = 42
	48 in. = 48
	54 in. = 54
	60 in. = 60
	72 in. = 72
	78 in. = 78
	84 in. = 84
	96 in. = 96
	108 in. = 108
	120 in. = 120
	144 in. = 144

Parameters inherited from **`trenched-conduit-template`**.
`backfill, construction-year, cover-depth, dewatering, easment, existing-utilities, length, manhole-spacing, pavement-restoration, row, traffic, trench-safety, type`

Cross Reference Table Definitions

Cross Reference Tables inherited from **`trenched-conduit-template`**.
`average-traffic-unitcost, average-utility-unitcost, complex-utility-unitcost, concrete-material, concrete-odiam, dewatering-well-unitcost, force-main-material, force-main-odiam, heavy-traffic-unitcost, high-force-main-material, installation-cost, manhole-additional-cost, manhole-base-cost, pavement-restoration-width, row-unitcost, trench-sump-dewatering-unitcost`

Calculator Definition

Description: *Parallel Pipe 1.0*
Inherits from *Trenched Conduit Template 1.0* `(trenched-conduit-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`excavation-depth`	Excavation Depth	`ppipe`
`manhole-spacing`	Manhole Spacing	`trenched-conduit-template`
`min-easment-width`	Min. Easement Width	`trenched-conduit-template`
`min-pavement-width`	Complete Surface Rest. Width	`trenched-conduit-template`
`outer-diameter`	Outer Diameter 1	`ppipe`
`outer-diameter2`	Outer Diameter 2	`ppipe`
`trench-width`	Trench Width	`ppipe`

Line Item Cost List		
Name	**Description**	**Defined In**
`backfill-cost`	Backfill	`trenched-conduit-template`

`dewatering-cost`	Dewatering	`ppipe`
`easment-cost`	Easement	`trenched-conduit-template`
`excavation-cost`	Excavation	`trenched-conduit-template`
`manhole-cost`	Manholes	`ppipe`
`pipe-installation-cost`	Pipe Installation	`ppipe`
`pipe-material-cost`	Pipe Unit Material Cost	`ppipe`
`place-pipe-zone-fill-cost`	Place Pipe Zone Fill	`ppipe`
`restoration-overlay-cost`	Overlay Pavement Restoration	`trenched-conduit-template`
`restoration-trench-cost`	Complete Pavement Restoration	`trenched-conduit-template`
`row-cost`	Land Acquisition	`trenched-conduit-template`
`spoil-load-and-haul-cost`	Spoil Load and Haul	`ppipe`
`traffic-control-cost`	Traffic Control	`ppipe`
`trench-safety-cost`	Trench Safety	`trenched-conduit-template`
`utilities-cost`	Existing Utilities	`ppipe`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenched-conduit-template`

Variable Definitions

excavation-depth
Category: *Geometry* Description: *Excavation Depth* Units: `ft`
Variable = ({`cover-depth`} + *max*[`<outer-diameter>`, `<outer-diameter2>`] + **1**)

outer-diameter
Category: *Geometry* Description: *Outer Diameter 1* Units: `ft`
Variable = ([*if*({`type`}=**"gravity"**, [*lookup* {`diameter1`} in **[`concrete-odiam`]**, **0**) + *if*({`type`}=**"force"**, [*lookup* {`diameter1`} in **[`force-main-odiam`]**, **0**) + *if*({`type`}=**"high-force"**, [*lookup* {`diameter1`} in **[`force-main-odiam`]**, **0**)] X [**1**/ **12**])

outer-diameter2
Category: *Geometry* Description: *Outer Diameter 2* Units: `ft`
Variable = ([*if*({`type`}=**"gravity"**, [*lookup* {`diameter2`} in **[`concrete-odiam`]**, **0**) + *if*({`type`}=**"force"**, [*lookup* {`diameter2`} in **[`force-main-odiam`]**, **0**) + *if*({`type`}=**"high-force"**, [*lookup* {`diameter2`} in **[`force-main-odiam`]**, **0**)] X [**1**/ **12**])

trench-width
Category: *Geometry* Description: *Trench Width* Units: `ft`
Variable = ([(`<outer-diameter>` + `<outer-diameter2>` + **1** + [(`<outer-diameter>` + `<outer-diameter2>`) X **0.15**]) X **1.25**] + **1**)

Variables inherited from **`trenched-conduit-template`**.
`manhole-spacing, min-easment-width, min-pavement-width`

Line Item Cost Definitions

dewatering-cost
Description: *Dewatering* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`dewatering`}=**"minimal"**, *max*([*lookup* {`diameter1`} in **[`trench-sump-dewatering-unitcost`]**], [*lookup* {`diameter2`} in **[`trench-sump-dewatering-unitcost`]**]), **0**] + *if*[{`dewatering`}=**"significant"**, *max*([*lookup* {`diameter1`} in **[`dewatering-well-unitcost`]**], [*lookup* {`diameter2`} in **[`dewatering-well-unitcost`]**]), **0**])

manhole-cost
Description: *Manholes* Unit Value: `MH`
Quantity = *if*(<`manhole-spacing`> > **0**, [*ceiling*([{`length`} X (**1**/ {`manhole-spacing`})]) X **2**], **0**)
Unit Cost = ([(*lookup* {`diameter1`} in **[`manhole-base-cost`]**) + (*lookup* {`diameter2`} in **[`manhole-base-cost`]**) + ([(*lookup* {`diameter1`} in **[`manhole-additional-cost`]**) + (*lookup* {`diameter2`} in **[`manhole-additional-cost`]**)] X *max*[({`cover-depth`} + **-12**), **0**])] X **0.5**)

pipe-installation-cost
Description: *Pipe Installation* Unit Value: `lf`

Quantity = {`length`}
Unit Cost = ([*lookup* {`diameter1`} in **[`installation-cost`]** + [*lookup* {`diameter2`} in **[`installation-cost`]**])

pipe-material-cost
Description: *Pipe Unit Material Cost* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`type`}=**"force"**, ([*lookup* {`diameter1`} in **[`force-main-material`]** + [*lookup* {`diameter2`} in **[`force-main-material`]**]), **0**] + *if*[{`type`}=**"high-force"**, ([*lookup* {`diameter1`} in **[`high-force-main-material`]** + [*lookup* {`diameter2`} in **[`high-force-main-material`]**]), **0**] + *if*[{`type`}=**"gravity"**, ([*lookup* {`diameter1`} in **[`concrete-material`]** + [*lookup* {`diameter2`} in **[`concrete-material`]**]), **0**])

place-pipe-zone-fill-cost
Description: *Place Pipe Zone Fill* Unit Value: `CY`
Quantity = ([([*max*(`<outer-diameter>`, `<outer-diameter2>`) + **2**] X `<trench-width>`) + (**-3.141592654** X `<outer-diameter>` X `<outer-diameter>` X **0.25**) + (**-3.141592654** X `<outer-diameter2>` X `<outer-diameter2>` X **0.25**)] X {`length`} X [**1**/ **27**])
Unit Cost = **{`import-fill-unitcost`}**

spoil-load-and-haul-cost
Description: *Spoil Load and Haul* Unit Value: `CY`
Quantity = ([*max*(`<outer-diameter>`, `<outer-diameter2>`) + **2** + *if*({`backfill`}=**"imported"**, *max*[**0**, ({`cover-depth`} + **-1**)], **0**)] X `<trench-width>` X {`length`} X [**1**/ **27**])

Unit Cost = **{spoil-load-and-haul-unitcost}**

traffic-control-cost
Description: *Traffic Control* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`traffic`}=**"light"**, *max*([*lookup* {`diameter1`} in **[average-traffic-unitcost]**], [*lookup* {`diameter2`} in **[average-traffic-unitcost]**]), **0**] + *if*[{`traffic`}=**"heavy"**, *max*([*lookup* {`diameter1`} in **[heavy-traffic-unitcost]**], [*lookup* {`diameter2`} in **[heavy-traffic-unitcost]**]), **0**])

utilities-cost
Description: *Existing Utilities* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`existing-utilities`}=**"average"**, *max*([*lookup* {`diameter1`} in **[average-utility-unitcost]**], [*lookup* {`diameter2`} in **[average-utility-unitcost]**]), **0**] + *if*[{`existing-utilities`}=**"complex"**, *max*([*lookup* {`diameter1`} in **[complex-utility-unitcost]**], [*lookup* {`diameter2`} in **[complex-utility-unitcost]**]), **0**])

Line item costs inherited from **`trenched-conduit-template`**.
`backfill-cost, easment-cost, excavation-cost, restoration-overlay-cost, restoration-trench-cost, row-cost, trench-safety-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenched-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = {`construction-year`}

Pipe

Programatic name: **`pipe`**
Specialized User Interface: `tabula.model.pipe.PipeInfoFrame`
Customized Icon Path: `tabula/model/pipe/tree_icon.gif`

Domain Definition

Description: *Pipe*
Inherits from *Trenched Conduit Template* `(trenched-conduit-template)`

Summary

Constant List	
Name	**Defined In**
`asphalt-paving-unitcost`	`trenched-conduit-template`
`excavation-unitcost`	`trenched-conduit-template`
`import-fill-unitcost`	`trenched-conduit-template`
`native-fill-unitcost`	`trenched-conduit-template`
`overlay-paving-unitcost`	`trenched-conduit-template`
`special-shoring-unitcost`	`trenched-conduit-template`
`spoil-load-and-haul-unitcost`	`trenched-conduit-template`
`trench-safety-unitcost`	`trenched-conduit-template`

Parameter List		
Name	**Description**	**Defined In**
`backfill`	Trench Backfill Type	`trenched-conduit-template`
`construction-year`	Construction Year	`trenched-conduit-template`
`cover-depth`	Depth of Cover	`trenched-conduit-template`
`dewatering`	Dewatering	`trenched-conduit-template`
`diameter`	Pipe Diameter	`pipe`
`easment`	Required Easements	`trenched-conduit-template`
`existing-utilities`	Existing Utilities	`trenched-conduit-template`
`length`	Length	`trenched-conduit-template`
`manhole-spacing`	Manhole Spacing	`trenched-conduit-template`
`pavement-restoration`	Pavement Restoration	`trenched-conduit-template`
`row`	Land Acquisition	`trenched-conduit-template`
`traffic`	Traffic	`trenched-conduit-template`
`trench-safety`	Trench Safety	`trenched-conduit-template`
`type`	Conduit Type	`trenched-conduit-template`

Cross Reference Lookup Table List	
Name	**Defined In**
`average-traffic-unitcost`	`trenched-conduit-template`
`average-utility-unitcost`	`trenched-conduit-template`
`complex-utility-unitcost`	`trenched-conduit-template`
`concrete-material`	`trenched-conduit-template`
`concrete-odiam`	`trenched-conduit-template`
`dewatering-well-unitcost`	`trenched-conduit-template`

force-main-material	trenched-conduit-template
force-main-odiam	trenched-conduit-template
heavy-traffic-unitcost	trenched-conduit-template
high-force-main-material	trenched-conduit-template
installation-cost	trenched-conduit-template
manhole-additional-cost	trenched-conduit-template
manhole-base-cost	trenched-conduit-template
pavement-restoration-width	trenched-conduit-template
row-unitcost	trenched-conduit-template
trench-sump-dewatering-unitcost	trenched-conduit-template

Constant Definitions

No Defined Constants

Constants inherited from **trenched-conduit-template**.
asphalt-paving-unitcost, excavation-unitcost, import-fill-unitcost, native-fill-unitcost, overlay-paving-unitcost, special-shoring-unitcost, spoil-load-and-haul-unitcost, trench-safety-unitcost

Parameter Definitions

Name	Description	Type	Default	Units
diameter	Pipe Diameter	enum	12	n/a
			Valid Values:	8 in. = 8
				10 in. = 10
				12 in. = 12
				14 in. = 14
				15 in. = 15
				16 in. = 16

18 in. = 18
20 in. = 20
21 in. = 21
24 in. = 24
27 in. = 27
30 in. = 30
36 in. = 36
42 in. = 42
48 in. = 48
54 in. = 54
60 in. = 60
72 in. = 72
78 in. = 78
84 in. = 84
96 in. = 96
108 in. = 108
120 in. = 120
144 in. = 144

Parameters inherited from **`trenched-conduit-template`**.
`backfill, construction-year, cover-depth, dewatering, easment, existing-utilities, length, manhole-spacing, pavement-restoration, row, traffic, trench-safety, type`

Cross Reference Table Definitions

Cross Reference Tables inherited from **`trenched-conduit-template`**.
`average-traffic-unitcost, average-utility-unitcost, complex-utility-unitcost, concrete-material, concrete-odiam, dewatering-well-unitcost, force-main-material, force-main-odiam, heavy-traffic-unitcost, high-force-main-material, installation-cost, manhole-additional-cost, manhole-base-cost, pavement-restoration-width, row-unitcost, trench-sump-dewatering-unitcost`

Calculator Definition

Description: *Pipe 1.0*
Inherits from *Trenched Conduit Template 1.0* `(trenched-conduit-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`excavation-depth`	Excavation Depth	`pipe`
`manhole-spacing`	Manhole Spacing	`trenched-conduit-template`
`min-easment-width`	Min. Easement Width	`trenched-conduit-template`
`min-pavement-width`	Complete Surface Rest. Width	`trenched-conduit-template`
`outer-diameter`	Outer Diameter	`pipe`
`outer-diameter2`	Outer Diameter 2 Placeholder	`trenched-conduit-template`
`trench-width`	Trench Width	`pipe`

Line Item Cost List		
Name	**Description**	**Defined In**
`backfill-cost`	Backfill	`trenched-conduit-template`
`dewatering-cost`	Dewatering	`pipe`
`easment-cost`	Easement	`trenched-conduit-template`
`excavation-cost`	Excavation	`trenched-conduit-template`
`manhole-cost`	Manholes	`pipe`
`pipe-installation-cost`	Pipe Installation	`pipe`

`pipe-material-cost`	Pipe Unit Material Cost	`pipe`
`place-pipe-zone-fill-cost`	Place Pipe Zone Fill	`pipe`
`restoration-overlay-cost`	Overlay Pavement Restoration	`trenched-conduit-template`
`restoration-trench-cost`	Complete Pavement Restoration	`trenched-conduit-template`
`row-cost`	Land Acquisition	`trenched-conduit-template`
`spoil-load-and-haul-cost`	Spoil Load and Haul	`pipe`
`traffic-control-cost`	Traffic Control	`pipe`
`trench-safety-cost`	Trench Safety	`trenched-conduit-template`
`utilities-cost`	Existing Utilities	`pipe`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenched-conduit-template`

Variable Definitions

excavation-depth
Category: *Geometry* Description: *Excavation Depth* Units: `ft`
Variable = ({`cover-depth`} + `<outer-diameter>` + **1**)

outer-diameter
Category: *Geometry* Description: *Outer Diameter* Units: `ft`
Variable = ([*if*({`type`}="**gravity**", [*lookup* {`diameter`} in **[`concrete-odiam`]**], **0**) + *if*({`type`}="**force**", [*lookup* {`diameter`} in **[`force-main-odiam`]**], **0**) + *if*({`type`}="**high-force**", [*lookup* {`diameter`} in **[`force-main-odiam`]**], **0**)] X [**1**/ **12**])

trench-width
Category: *Geometry* Description: *Trench Width* Units: `ft`
Variable = ([**1.3** X <outer-diameter>] + **2.5**)

Variables inherited from **`trenched-conduit-template`**.
`manhole-spacing, min-easment-width, min-pavement-width, outer-diameter2`

Line Item Cost Definitions

dewatering-cost
Description: *Dewatering* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`dewatering`}="**minimal**", (*lookup* {`diameter`} in **[`trench-sump-dewatering-unitcost`]**), **0**] + *if*[{`dewatering`}="**significant**", (*lookup* {`diameter`} in **[`dewatering-well-unitcost`]**), **0**])

manhole-cost

Description: *Manholes* Unit Value: `MH`
Quantity = *if*(`<manhole-spacing>` > **0**, *ceiling*[({`length`} `X` [**1**/ {`manhole-spacing`}])], **0**)
Unit Cost = ([*lookup* {`diameter`} in **[`manhole-base-cost`]** `+` [(*lookup* {`diameter`} in **[`manhole-additional-cost`]**) `X` *max*([{`cover-depth`} `+` **-12**], **0**)])

pipe-installation-cost
Description: *Pipe Installation* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*lookup* {`diameter`} in **[`installation-cost`]**)

pipe-material-cost
Description: *Pipe Unit Material Cost* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`type`}=**"force"**, (*lookup* {`diameter`} in **[`force-main-material`]**), **0**] `+` *if*[{`type`}=**"high-force"**, (*lookup* {`diameter`} in **[`high-force-main-material`]**), **0**] `+` *if*[{`type`}=**"gravity"**, (*lookup* {`diameter`} in **[`concrete-material`]**), **0**])

place-pipe-zone-fill-cost
Description: *Place Pipe Zone Fill* Unit Value: `CY`
Quantity = ([([`<outer-diameter>` `+` **2**] `X` `<trench-width>`) `+` (**-3.141592654** `X` `<outer-diameter>` `X` `<outer-diameter>` `X` **0.25**)] `X` {`length`} `X` [**1**/ **27**])
Unit Cost = **{`import-fill-unitcost`}**

spoil-load-and-haul-cost
Description: *Spoil Load and Haul* Unit Value: `CY`
Quantity = ([`<outer-diameter>` + **2** + *if*({`backfill`}="**imported**", *max*[**0**, ({`cover-depth`} + **-1**)], **0**)] X `<trench-width>` X {`length`} X [**1**/ **27**])
Unit Cost = **{`spoil-load-and-haul-unitcost`}**

traffic-control-cost
Description: *Traffic Control* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`traffic`}="**light**", (*lookup* {`diameter`} in **[`average-traffic-unitcost`]**), **0**] + *if*[{`traffic`}="**heavy**", (*lookup* {`diameter`} in **[`heavy-traffic-unitcost`]**), **0**])

utilities-cost
Description: *Existing Utilities* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`existing-utilities`}="**average**", (*lookup* {`diameter`} in **[`average-utility-unitcost`]**), **0**] + *if*[{`existing-utilities`}="**complex**", (*lookup* {`diameter`} in **[`complex-utility-unitcost`]**), **0**])

Line item costs inherited from **`trenched-conduit-template`**.
`backfill-cost, easment-cost, excavation-cost, restoration-overlay-cost, restoration-trench-cost, row-cost, trench-safety-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenched-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = {`construction-year`}

Project

Programatic name: **`project`**
Specialized User Interface: `tabula.model.project.ProjectInfoFrame`
Customized Icon Path: `tabula/model/project/tree_icon.gif`

Domain Definition

Description: *Project*

Summary

Parameter List		
Name	**Description**	**Defined In**
`comments`	Comments	`project`
`project-year`	Project Year	`project`

Constant Definitions

No Defined Constants

Parameter Definitions

Name	Description	Type	Default	Units
`comments`	Comments	comment		n/a
`project-year`	Project Year	year	2006	n/a

Cross Reference Table Definitions

Calculator Definition

Description: *Project*
Basis Year: **`2005`**

Summary

Variable Definitions

No Defined Variables

Line Item Cost Definitions

No Defined Line Item Costs

Multiplier Definitions

No Defined Multipliers

Year Definition

Year Calculation
Year = `{project-year}`

Pump Station

Programatic name: **`pump`**
Specialized User Interface: `tabula.model.pump.PumpInfoFrame`
Customized Icon Path: `tabula/model/pump/tree_icon.gif`

Domain Definition

Description: *Pump Station*

Summary

Parameter List		
Name	**Description**	**Defined In**
`construction-year`	Construction Year	`pump`
`excavation-depth`	Excavation Depth	`pump`
`firm-capacity`	Firm Capacity	`pump`
`tdh`	Total Dynamic Head	`pump`

Constant Definitions

No Defined Constants

Parameter Definitions

Name	Description	Type	Default	Units
`construction-year`	Construction Year	year	2006	n/a
`excavation-depth`	Excavation Depth	length	30	ft
`firm-capacity`	Firm Capacity	rate	20	mgd
		(Valid values range from 0.5 to 80.)		
`tdh`	Total Dynamic Head	length	50	ft
		(Valid values range from 20 to 400.)		

Cross Reference Table Definitions

Calculator Definition

Description: *Pump Station 1.0*
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`architectural-adjustment-unitcost`	Architectural/Structural Unit Cost Adjustment	`pump`
`base-architectural-unitcost`	Base Architectural/Structural Unit Cost	`pump`
`base-mechanical-unitcost`	Base Mechanical Unit Cost	`pump`
`mechanical-adjustment-unitcost`	Mechanical Unit Cost Adjustment	`pump`
`power`	Required Pump Power	`pump`
`tdh`	TDH	`pump`

Line Item Cost List		
Name	**Description**	**Defined In**
`architectural-cost`	Architectural/Structural	`pump`
`electrical-cost`	Electrical/Instrumentation	`pump`
`mechanical-cost`	Mechanical	`pump`
`site-cost`	Site/Civil	`pump`

Variable Definitions

architectural-adjustment-unitcost
Category: *Calculated Parameters* Description: *Architectural/Structural Unit Cost Adjustment* Units: `$/mgd`
Variable = (**0.24** x [([{`excavation-depth`} + **-30**] x [**1**/ **30**]) + (**0.01** x [{`tdh`} + **-120**] x [**1**/ **120**])] x **1000000**)

base-architectural-unitcost
Category: *Calculated Parameters* Description: *Base Architectural/Structural Unit Cost* Units: `$/mgd`
Variable = [(**40.29** X [{`firm-capacity`}]^{2}) – (**3597** x {`firm-capacity`}) + **206344**)]

base-mechanical-unitcost
Category: *Calculated Parameters* Description: *Base Mechanical Unit Cost* Units: `$/mgd`
Variable = (**261700** X [{`firm-capacity`}]$^{-0.289}$)

mechanical-adjustment-unitcost
Category: *Calculated Parameters* Description: *Mechanical Unit Cost Adjustment* Units: `$/mgd`
Variable = ([*if*(<tdh> <= **300**, [**0.06** X ({tdh} + **-120**) X (**1**/ **120**) X **1000000**], **0**) + *if*(<tdh> > **300**, [**1.2** X <base-mechanical-unitcost>], **0**)])

power
Category: *Calculated Parameters* Description: *Required Pump Power* Units: `Hp`
Variable = (**62.4** X {`firm-capacity`} X **1.547** X {tdh} X [**1**/ (**0.6** X **0.85** X **550**)])

tdh
Description: *TDH* Units: `ft`
Variable = {`tdh`}

Line Item Cost Definitions

architectural-cost
Description: *Architectural/Structural* Unit Value: `mgd`
Quantity = {`firm-capacity`}
Unit Cost = (`<base-architectural-unitcost> + <architectural-adjustment-unitcost>`)

electrical-cost
Description: *Electrical/Instrumentation* Unit Value: `LS`
Quantity = **1**
Unit Cost = (**12000** `X` [`<power>`]$^{0.70}$)

mechanical-cost
Description: *Mechanical* Unit Value: `mgd`
Quantity = {`firm-capacity`}
Unit Cost = (`<base-mechanical-unitcost> + <mechanical-adjustment-unitcost>`)

site-cost
Description: *Site/Civil* Unit Value: `LS`
Quantity = **1**
Unit Cost = (**57000** x [{`firm-capacity`}]$^{0.90}$)

Multiplier Definitions

No Defined Multipliers

Year Definition

Year Calculation
Year = {`construction-year`}

Reclaimed Water

Programatic name: **`rcw`**
Specialized User Interface: `tabula.model.rcw.RCWInfoFrame`
Customized Icon Path: `tabula/model/rcw/tree_icon.gif`

Domain Definition

Description: *Reclaimed Water System*
Inherits from *Trenched Conduit Template* `(trenched-conduit-template)`

Summary

Constant List	
Name	**Defined In**
`asphalt-paving-unitcost`	`trenched-conduit-template`
`excavation-unitcost`	`trenched-conduit-template`
`import-fill-unitcost`	`trenched-conduit-template`
`native-fill-unitcost`	`trenched-conduit-template`

overlay-paving-unitcost	trenched-conduit-template
special-shoring-unitcost	trenched-conduit-template
spoil-load-and-haul-unitcost	trenched-conduit-template
trench-safety-unitcost	trenched-conduit-template

Parameter List

Name	Description	Defined In
backfill	Trench Backfill Type	trenched-conduit-template
construction-year	Construction Year	trenched-conduit-template
cover-depth	Depth of Cover	trenched-conduit-template
dewatering	Dewatering	trenched-conduit-template
easment	Required Easements	trenched-conduit-template
existing-utilities	Existing Utilities	trenched-conduit-template
length	Length	trenched-conduit-template
manhole-spacing	Manhole Spacing	trenched-conduit-template
meter-count	Meter Count	rcw
pavement-restoration	Pavement Restoration	trenched-conduit-template
rcw-diameter	Pipe Diameter	rcw
row	Land Acquisition	trenched-conduit-template
traffic	Traffic	trenched-conduit-template
trench-safety	Trench Safety	trenched-conduit-template
type	Conduit Type	trenched-conduit-template

Cross Reference Lookup Table List

Name	Defined In

average-traffic-unitcost	trenched-conduit-template
average-utility-unitcost	trenched-conduit-template
complex-utility-unitcost	trenched-conduit-template
concrete-material	trenched-conduit-template
concrete-odiam	trenched-conduit-template
dewatering-well-unitcost	trenched-conduit-template
force-main-material	trenched-conduit-template
force-main-odiam	trenched-conduit-template
heavy-traffic-unitcost	trenched-conduit-template
high-force-main-material	trenched-conduit-template
installation-cost	trenched-conduit-template
manhole-additional-cost	trenched-conduit-template
manhole-base-cost	trenched-conduit-template
pavement-restoration-width	trenched-conduit-template
row-unitcost	trenched-conduit-template
trench-sump-dewatering-unitcost	trenched-conduit-template

Constant Definitions

No Defined Constants

Constants inherited from **trenched-conduit-template**.
asphalt-paving-unitcost, excavation-unitcost, import-fill-unitcost, native-fill-unitcost, overlay-paving-unitcost, special-shoring-unitcost, spoil-load-and-haul-unitcost, trench-safety-unitcost

Parameter Definitions

Name	Description	Type	Default	Units
rcw-diameter	Pipe Diameter	enum	8	n/a

Valid Values:	2 in. = 2
	10 in. = 10
	12 in. = 12
	14 in. = 14
	15 in. = 15
	16 in. = 16
	18 in. = 18
	20 in. = 20
	21 in. = 21
	24 in. = 24
	27 in. = 27
	30 in. = 30
	36 in. = 36
	42 in. = 42
	48 in. = 48
	54 in. = 54
	60 in. = 60
	72 in. = 72
	78 in. = 78
	84 in. = 84
	96 in. = 96
	108 in. = 108
	120 in. = 120
	144 in. = 144

Parameters inherited from **`trenched-conduit-template`**.
`backfill, construction-year, cover-depth, dewatering, easment, existing-utilities, length, manhole-spacing, pavement-restoration, row, traffic, trench-safety, type`

Cross Reference Table Definitions

Cross Reference Tables inherited from **`trenched-conduit-template`**.
`average-traffic-unitcost, average-utility-unitcost, complex-utility-unitcost, concrete-material, concrete-odiam, dewatering-well-unitcost, force-main-material, force-main-odiam, heavy-traffic-unitcost, high-force-main-material, installation-cost, manhole-additional-cost, manhole-base-cost, pavement-restoration-width, row-unitcost, trench-sump-dewatering-unitcost`

Calculator Definition

Description: *Reclaimed Water 1.0*
Inherits from *Trenched Conduit Template 1.0* `(trenched-conduit-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`excavation-depth`	Excavation Depth	`pipe`
`manhole-spacing`	Manhole Spacing	`trenched-conduit-template`
`min-easment-width`	Min. Easement Width	`trenched-conduit-template`
`min-pavement-width`	Complete Surface Rest. Width	`trenched-conduit-template`
`outer-diameter`	Outer Diameter	`pipe`
`outer-diameter2`	Outer Diameter 2 Placeholder	`trenched-conduit-template`
`trench-width`	Trench Width	`pipe`

Line Item Cost List		
Name	**Description**	**Defined In**
`backfill-cost`	Backfill	`trenched-conduit-template`

`dewatering-cost`	Dewatering	`pipe`
`easement-adjust`	Easement Adjustment Factor	`Rcw`
`easment-cost`	Easement	`trenched-conduit-template`
`excavation-cost`	Excavation	`trenched-conduit-template`
`manhole-cost`	Manholes	`pipe`
`pipe-installation-cost`	Pipe Installation	`pipe`
`pipe-material-cost`	Pipe Unit Material Cost	`pipe`
`place-pipe-zone-fill-cost`	Place Pipe Zone Fill	`pipe`
`restoration-overlay-cost`	Overlay Pavement Restoration	`trenched-conduit-template`
`restoration-trench-cost`	Complete Pavement Restoration	`trenched-conduit-template`
`row-cost`	Land Acquisition	`trenched-conduit-template`
`spoil-load-and-haul-cost`	Spoil Load and Haul	`pipe`
`traffic-control-cost`	Traffic Control	`pipe`
`trench-safety-cost`	Trench Safety	`trenched-conduit-template`
`utilities-cost`	Existing Utilities	`pipe`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenched-conduit-template`

Variable Definitions

excavation-depth
Category: *Geometry* Description: *Excavation Depth* Units: `ft`
Variable = ({`cover-depth`} + `<outer-diameter>` + **1**)

outer-diameter
Category: *Geometry* Description: *Outer Diameter* Units: `ft`
Variable = ([*if*({`type`}="**gravity**", [*lookup* {`diameter`} in **[`concrete-odiam`]**, **0**) + *if*({`type`}="**force**", [*lookup* {`diameter`} in **[`force-main-odiam`]**, **0**) + *if*({`type`}="**high-force**", [*lookup* {`diameter`} in **[`force-main-odiam`]**, **0**)] X [**1**/ **12**])

trench-width
Category: *Geometry* Description: *Trench Width* Units: `ft`
Variable = ([**1.3** X `<outer-diameter>`] + **2.5**)

Variables inherited from **`trenched-conduit-template`**.
`manhole-spacing, min-easment-width, min-pavement-width, outer-diameter2`

Line Item Cost Definitions

dewatering-cost
Description: *Dewatering*

Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`dewatering`}=**"minimal"**, (*lookup* {`diameter`} in **[`trench-sump-dewatering-unitcost`]**), **0**] + *if*[{`dewatering`}=**"significant"**, (*lookup* {`diameter`} in **[`dewatering-well-unitcost`]**), **0**])

manhole-cost
Description: *Manholes* Unit Value: `MH`
Quantity = *if*(<`manhole-spacing`> > **0**, *ceiling*[({`length`} X [**1**/ {`manhole-spacing`}])], **0**)
Unit Cost = ([*lookup* {`diameter`} in **[`manhole-base-cost`]**] + [(*lookup* {`diameter`} in **[`manhole-additional-cost`]**) X *max*([{`cover-depth`} + **-12**], **0**)])

pipe-installation-cost
Description: *Pipe Installation* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*lookup* {`diameter`} in **[`installation-cost`]**)

pipe-material-cost
Description: *Pipe Unit Material Cost* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`type`}=**"force"**, (*lookup* {`diameter`} in **[`force-main-material`]**), **0**] + *if*[{`type`}=**"high-force"**, (*lookup* {`diameter`} in **[`high-force-main-material`]**), **0**] + *if*[{`type`}=**"gravity"**, (*lookup* {`diameter`} in **[`concrete-material`]**), **0**])

place-pipe-zone-fill-cost
Description: *Place Pipe Zone Fill* Unit Value: `CY`
Quantity = ([([`<outer-diameter>` + **2**] X `<trench-width>`) + (**-3.141592654** X `<outer-diameter>` X `<outer-diameter>` X **0.25**)] X {`length`} X [**1**/ **27**])
Unit Cost = **{`import-fill-unitcost`}**

spoil-load-and-haul-cost
Description: *Spoil Load and Haul* Unit Value: `CY`
Quantity = ([`<outer-diameter>` + **2** + *if*({`backfill`}=**"imported"**, *max*[**0**, ({`cover-depth`} + **-1**)], **0**)] X `<trench-width>` X {`length`} X [**1**/ **27**])
Unit Cost = **{`spoil-load-and-haul-unitcost`}**

traffic-control-cost
Description: *Traffic Control* Unit Value: `lf`
Quantity = {`length`}
Unit Cost = (*if*[{`traffic`}=**"light"**, (*lookup* {`diameter`} in **[`average-traffic-unitcost`]**), **0**] + *if*[{`traffic`}=**"heavy"**, (*lookup* {`diameter`} in **[`heavy-traffic-unitcost`]**), **0**])

utilities-cost
Description: *Existing Utilities* Unit Value: `lf`
Quantity = {`length`}

Unit Cost = (*if*[{`existing-utilities`}=**"average"**, (*lookup* {`diameter`} in **[`average-utility-unitcost`]**), **0**] + *if*[{`existing-utilities`}=**"complex"**, (*lookup* {`diameter`} in **[`complex-utility-unitcost`]**), **0**])

Line item costs inherited from **`trenched-conduit-template`**.
`backfill-cost, easment-cost, excavation-cost, restoration-overlay-cost, restoration-trench-cost, row-cost, trench-safety-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenched-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation
Year = {`construction-year`}

Storage Facility

Programatic name: **`storage`**
Specialized User Interface: `tabula.model.storage.StorageInfoFrame`
Customized Icon Path: `tabula/model/storage/tree_icon.gif`

Domain Definition

Description: *Storage Facility*

Summary

Parameter List

Name	Description	Defined In
`construction-year`	Construction Year	`storage`
`dewatering`	Dewatering	`storage`
`footprint`	Facility Footprint	`storage`
`land-aquisition-type`	Land Acquisition	`storage`
`odor-control`	Odor Control	`storage`
`outflow-operations`	Outflow Operations	`storage`
`storage-capacity`	Storage Capacity	`storage`
`surface-restoration`	Surface Restoration	`storage`

Cross Reference Lookup Table List	
Name	**Defined In**
`property-row-unitcost`	`storage`
`surface-restoration-unitcost`	`storage`

Constant Definitions

No Defined Constants

Parameter Definitions

Name	Description	Type	Default	Units
`construction-year`	Construction Year	year	2006	n/a
`dewatering`	Dewatering	enum	standard	n/a
		Valid Values:	None = none Standard = standard Significant = significant	
`footprint`	Facility Footprint	area	100	SF

`land-aquisition-type`	Land Acquisition	enum	none	n/a
	Valid Values:	None = none Residential-Urban = residential Residential-Suburban = residential-suburban Residential-Rural = residential-rural Industrial = industrial Office/Commercial = commercial		
`odor-control`	Odor Control	boolean	false	n/a
`outflow-operations`	Outflow Operations	enum	pump	n/a
		Valid Values:	Gravity = gravity Pump = pump	
`storage-capacity`	Storage Capacity	volume	20	Mgal
		(Valid values range from 0.5 to 35.)		
`surface-restoration`	Surface Restoration	enum	hydroseed	n/a
		Valid Values:	None = none Hydroseed = hydroseed Pavement = pavement	

Cross Reference Table Definitions

property-row-unitcost	
Value (enum)	**Result** ($/sf)
`none`	0
`residential`	58
`residential-suburban`	36
`residential-rural`	24
`industrial`	23
`commercial`	24

surface-restoration-unitcost	
Value (enum)	**Result** ($/SY)
`none`	0
`hydroseed`	5
`pavement`	40

Calculator Definition

Description: *Storage Facility 1.0*
Basis Year: **`2005`**

Summary

Line Item Cost List		
Name	**Description**	**Defined In**
`construction-cost`	Construction Cost	`storage`
`dewatering-cost`	Dewatering	`storage`
`easment-cost`	Land Acquisition	`storage`
`effluent-pump-cost`	Effluent Pump Station	`storage`
`odor-control-cost`	Odor Control	`storage`
`surface-restoration-cost`	Surface Restoration	`storage`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`storage`

Variable Definitions

No Defined Variables

Line Item Cost Definitions

construction-cost
Description: *Construction Cost* Unit Value: `Mgal`
Quantity = {`storage-capacity`}
Unit Cost = (**1000000** `X` **15.02** `X` [{`storage-capacity`}]$^{-0.621}$)

dewatering-cost
Description: *Dewatering* Unit Value: `LS`
Quantity = **1**
Unit Cost = (*if*[{`dewatering`}=**"standard"**, ([**880** `X` {`storage-capacity`} `X` {`storage-capacity`}] + [**43000** `X` {`storage-capacity`}] + **400000**), **0**] + *if*[{`dewatering`}=**"significant"**, ([**1175** `X` {`storage-capacity`} `X` {`storage-capacity`}] + [**80500** `X` {`storage-capacity`}] + **765000**), **0**])

easment-cost
Description: *Land Acquisition* Unit Value: `SF`
Quantity = {`footprint`}
Unit Cost = (*lookup* {`land-aquisition-type`} in **[`property-row-unitcost`]**)

effluent-pump-cost
Description: *Effluent Pump Station*

Unit Value: `LS`
Quantity = **1**
Unit Cost = *if*({`outflow-operations`}=**"pump"**, [**1.35** `X` ([({`storage-capacity`})$^{0.85}$ `X` **22000**] `+` **120000**)], **0**)

odor-control-cost
Description: *Odor Control* Unit Value: `LS`
Quantity = **1**
Unit Cost = *if*({`odor-control`}=**"true"**, [**12000** `+` (**150000** `X` {`storage-capacity`})], **0**)

surface-restoration-cost
Description: *Surface Restoration* Unit Value: `SY`
Quantity = ({`footprint`} `X` [**1**/ **9**])
Unit Cost = (*lookup* {`surface-restoration`} in **[`surface-restoration-unitcost`]**)

Multiplier Definitions

mobilization-demobilization
Description: *Mobilization/Demobilization at 10 percent*
Multiplier = **1.1**

Year Definition

Year Calculation

Year = {`construction-year`}

Trenched Conduit Template

Programatic name: **`trenched-conduit-template`**
Not Public, hidden from user interface

Domain Definition

Description: *Trenched Conduit Template*

Summary

Constant List	
Name	**Defined In**
`asphalt-paving-unitcost`	`trenched-conduit-template`
`excavation-unitcost`	`trenched-conduit-template`
`import-fill-unitcost`	`trenched-conduit-template`
`native-fill-unitcost`	`trenched-conduit-template`
`overlay-paving-unitcost`	`trenched-conduit-template`
`special-shoring-unitcost`	`trenched-conduit-template`
`spoil-diposal-cost`	`trenched-conduit-template`
`spoil-load-and-haul-unitcost`	`trenched-conduit-template`
`trench-safety-unitcost`	`trenched-conduit-template`

Parameter List		
Name	**Description**	**Defined In**
`backfill`	Trench Backfill Type	`trenched-conduit-template`
`construction-year`	Construction Year	`trenched-conduit-template`
`cover-depth`	Depth of Cover	`trenched-conduit-template`

dewatering	Dewatering	trenched-conduit-template
easment	Required Easements	trenched-conduit-template
existing-utilities	Existing Utilities	trenched-conduit-template
length	Length	trenched-conduit-template
manhole-spacing	Manhole Spacing	trenched-conduit-template
pavement-restoration	Pavement Restoration	trenched-conduit-template
row	Land Acquisition	trenched-conduit-template
spoils-disposal	Disposal Type	trenched-conduit-template
traffic	Traffic	trenched-conduit-template
trench-safety	Trench Safety	trenched-conduit-template
type	Conduit Type	trenched-conduit-template

Cross Reference Lookup Table List	
Name	**Defined In**
average-traffic-unitcost	trenched-conduit-template
average-utility-unitcost	trenched-conduit-template
complex-utility-unitcost	trenched-conduit-template
concrete-material	trenched-conduit-template
concrete-odiam	trenched-conduit-template
dewatering-well-unitcost	trenched-conduit-template
disposal-unitcost	trenched-conduit-template
force-main-material	trenched-conduit-template
force-main-odiam	trenched-conduit-template
heavy-traffic-unitcost	trenched-conduit-template
high-force-main-material	trenched-conduit-template
installation-cost	trenched-conduit-template

manhole-additional-cost	trenched-conduit-template
manhole-base-cost	trenched-conduit-template
pavement-restoration-width	trenched-conduit-template
row-unitcost	trenched-conduit-template
trench-sump-dewatering-unitcost	trenched-conduit-template

Constant Definitions

Name	Type	Value	Units
asphalt-paving-unitcost	cost	55	$/SY
excavation-unitcost	cost	12	$/CY
import-fill-unitcost	cost	32	$/CY
native-fill-unitcost	cost	8	$/CY
overlay-paving-unitcost	cost	25	$/SY
special-shoring-unitcost	cost	12	$/SF
spoil-load-and-haul-unitcost	cost	12	$/CY
trench-safety-unitcost	cost	0.5	$/SF

Parameter Definitions

Name	Description	Type	Default	Units
backfill	Trench Backfill Type	enum	imported	n/a
		Valid Values:	Native = native Imported = imported	
construction-year	Construction Year	year	2006	n/a
cover-depth	Depth of Cover	decimal	12	ft
		(Valid values range from 4 to 40.)		

`dewatering`	Dewatering	enum	minimal	n/a
	Valid Values:	None = none Minimal = minimal Significant = significant		
`easment`	Required Easements	enum	none	n/a
	Valid Values:	None = none Residential = residential Industrial = industrial Office/Commercial = commercial		
`existing-utilities`	Existing Utilities	enum	complex	n/a
	Valid Values:	None = none Average = average Complex = complex		
`length`	Length	length	100	ft
`manhole-spacing`	Manhole Spacing	enum	500	ft
	Valid Values:	None = 0 Close (250 ft) = 250 Average (500 ft) = 500 Far (1000 ft) = 1000		
`pavement-restoration`	Pavement Restoration	enum	trench	n/a
	Valid Values:	None = none Trench Width = trench Half Width - Arterial (22 ft) = half Half Width - Collector Street (18 ft) = full-collector Half Width - Residential Street (14 ft) = full-residential Full Width - Arterial (44 ft) = full Full Width - Collector Street (36 ft) = full-collector Full Width - Residential Street (28 ft) = full-residential		
`row`	Land Acquisition	enum	none	n/a

		Valid Values:	None = none Residential-Urban = residential Residential-Suburban = residential-suburban Residential-Rural = residential-rural Industrial = industrial Office/Commercial = commercial	
`traffic`	Traffic	enum	heavy	n/a *Valid Values:* None = none Light = light Heavy = heavy
`trench-safety`	Trench Safety	enum	standard	n/a *Valid Values:* Standard = standard Special Shoring = special
`type`	Conduit Type	enum	gravity	n/a *Valid Values:* Gravity Sewer = gravity Force Main = force High Head Force Main = high-force

Cross Reference Table Definitions

average-traffic-unitcost	
Value (in)	**Result** ($/lf)
8	8
10	8
12	8
14	8
15	8
16	8

18	8
20	8
21	8
24	12
27	12
30	12
36	12
42	12
48	18
54	18
60	18
72	25
78	25
84	25
96	50
108	50
120	50
144	50

average-utility-unitcost	
Value (in)	**Result** ($/lf)
8	25
10	25
12	25
14	35
15	35

16	35
18	35
20	45
21	45
24	45
27	45
30	45
36	55
42	55
48	70
54	70
60	90
72	115
78	115
84	140
96	140
108	170
120	170
144	170

complex-utility-unitcost	
Value (in)	**Result** ($/lf)
8	45
10	45
12	45
14	70

15	70
16	70
18	70
20	90
21	90
24	90
27	90
30	90
36	115
42	115
48	140
54	140
60	185
72	230
78	230
84	275
96	275
108	345
120	345
144	345

concrete-material	
Value (in)	**Result** ($/lf)
8	8
10	10
12	16

14	NaN
15	17
16	NaN
18	22
20	NaN
21	24
24	32
27	49
30	60
36	73
42	95
48	121
54	151
60	210
72	270
78	320
84	380
96	490
108	610
120	740
144	1300

concrete-odiam	
Value (in)	**Result** (in)
8	10.5
10	12.5

12	17
14	NaN
15	20
16	NaN
18	23
20	NaN
21	26.5
24	30
27	33.5
30	37
36	44
42	51
48	58
54	66.5
60	73.5
72	87.5
78	96
84	100
96	115.5
108	128
120	140
144	168

dewatering-well-unitcost	
Value (in)	**Result** ($/lf)
8	60

10	60
12	60
14	65
15	65
16	65
18	65
20	65
21	65
24	75
27	75
30	75
36	80
42	80
48	80
54	95
60	95
66	95
72	110
78	110
84	110
90	125
96	125
108	150
120	150
144	150

force-main-material

Value (in)	**Result** ($/lf)
8	18
10	24
12	30
14	38
15	NaN
16	45
18	52
20	58
21	NaN
24	73
27	NaN
30	105
36	145
42	196
48	260
54	340
60	400
72	NaN
78	NaN
84	NaN
96	NaN
108	NaN
120	NaN
144	NaN

force-main-odiam	
Value (in)	**Result** (in)
8	9.05
10	11.1
12	13.2
14	15.3
15	NaN
16	17.4
18	19.5
20	21.6
21	NaN
24	25.8
27	NaN
30	32
36	38.3
42	44.5
48	50.8
54	57.76
60	61.61
72	NaN
78	NaN
84	NaN
96	NaN
108	NaN
120	NaN
144	NaN

heavy-traffic-unitcost	
Value (in)	**Result** ($/lf)
8	16
10	16
12	16
14	16
15	16
16	16
18	16
20	16
21	16
24	24
27	24
30	24
36	24
42	24
48	36
54	36
60	36
72	50
78	50
84	50
96	100
108	100
120	100
144	100

high-force-main-material	
Value (in)	**Result** ($/lf)
8	24
10	31
12	38
14	50
15	NaN
16	59
18	68
20	78
21	NaN
24	98
27	NaN
30	139
36	191
42	250
48	310
54	410
60	470
72	NaN
78	NaN
84	NaN
96	NaN
108	NaN
120	NaN
144	NaN

installation-cost	
Value (in)	**Result** ($/LF)
8	18
10	20
12	24
14	24
15	24
16	26
18	27
20	29
21	30
24	36
27	51
30	53
36	65
42	70
48	79
54	97
60	105
72	126
78	140
84	158
96	210
108	315
120	420
144	505

manhole-additional-cost	
Value (in)	**Result** ($/MH)
8	290
10	290
12	290
14	290
15	290
16	290
18	290
20	290
21	290
24	430
27	430
30	900
36	900
42	900
48	1300
54	1500
60	1500
72	1900
78	2200
84	2600
96	2600
108	2600
120	2600
144	2600

manhole-base-cost	
Value (in)	**Result** ($/MH)
8	4000
10	4000
12	4000
14	4000
15	4000
16	4000
18	4000
20	4000
21	4000
24	5000
27	5000
30	9500
36	9500
42	9500
48	14000
54	18000
60	18000
72	22000
78	29000
84	36000
96	36000
108	36000
120	36000
144	36000

pavement-restoration-width	
Value (enum)	**Result** (ft)
none	0
trench	0
half	22
full-collector	18
full-residential	14
full	44
full-collector	36
full-residential	28

row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	58
residential-suburban	36
residential-rural	24
industrial	23
commercial	37

disposal-unitcost	
Value (enum)	**Result** ($/cy)
No disposal cost	0
Common earth disposal	5

Contaminated earth disposal	225

trench-sump-dewatering-unitcost	
Value (in)	**Result** ($/lf)
8	20
10	20
12	20
14	20
15	20
16	20
18	20
20	20
21	20
24	20
27	20
30	20
36	30
42	30
48	30
54	30
60	30
66	45
72	45
78	45
84	45
90	55

96	55
108	75
120	75
144	75

Calculator Definition

Description: *Trenched Conduit Template 1.0*
Basis Year: **2005**

Summary

Variable List		
Name	**Description**	**Defined In**
excavation-depth	Excavation Depth Placeholder	trenched-conduit-template
manhole-spacing	Manhole Spacing	trenched-conduit-template
min-easment-width	Min. Easement Width	trenched-conduit-template
min-pavement-width	Complete Surface Rest. Width	trenched-conduit-template
outer-diameter	Outer Diameter 1 Placeholder	trenched-conduit-template
outer-diameter2	Outer Diameter 2 Placeholder	trenched-conduit-template
trench-width	Trench Width Placeholder	trenched-conduit-template

Line Item Cost List		
Name	**Description**	**Defined In**
backfill-cost	Backfill	trenched-conduit-template
easment-cost	Easement	trenched-conduit-template
excavation-cost	Excavation	trenched-conduit-template

`restoration-overlay-cost`	Overlay Pavement Restoration	`trenched-conduit-template`
`restoration-trench-cost`	Complete Pavement Restoration	`trenched-conduit-template`
`row-cost`	Land Acquisition	`trenched-conduit-template`
`trench-safety-cost`	Trench Safety	`trenched-conduit-template`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenched-conduit-template`

Variable Definitions

excavation-depth
Description: *Excavation Depth Placeholder* Units: `ft`
Variable = **0**

manhole-spacing
Description: *Manhole Spacing* Units: `ft`
Variable = *if*({`type`}=**"gravity"**, {`manhole-spacing`} , **0**)

min-easment-width

Description: *Min. Easement Width* Units: `ft`
Variable = (**20** + *if*[`<excavation-depth>` > **12**, **10**, **0**] + *if*[`<excavation-depth>` > **20**, **20**, **0**])

min-pavement-width
Category: *Geometry* Description: *Complete Surface Rest. Width* Units: `ft`
Variable = (*if*[{`pavement-restoration`}="**trench**", (`<trench-width>` + **2**), **0**] + *min*[(`<trench-width>` + **2**), (*lookup* {`pavement-restoration`} in **[`pavement-restoration-width`]**)])

outer-diameter
Description: *Outer Diameter 1 Placeholder* Units: `ft`
Variable = **0**

outer-diameter2
Description: *Outer Diameter 2 Placeholder* Units: `ft`
Variable = **0**

trench-width
Description: *Trench Width Placeholder* Units: `ft`

Variable = **0**

Line Item Cost Definitions

backfill-cost
Description: *Backfill* Unit Value: `CY`
Quantity = (`<trench-width>` `X` *max*[**0**, ({`cover-depth`} + **-1**)] `X` {`length`} `X` [**1**/ **27**])
Unit Cost = (*if*[{`backfill`}=**"imported"**, **{`import-fill-unitcost`}**, **0**] + *if*[{`backfill`}=**"native"**, **{`native-fill-unitcost`}**, **0**])

easment-cost
Description: *Easement* Unit Value: `SF`
Quantity = ({`length`} `X` `<min-easment-width>`)
Unit Cost = ([*lookup* {`easment`} in **[`row-unitcost`]**] `X` **0.3**)

excavation-cost
Description: *Excavation* Unit Value: `CY`
Quantity = (`<trench-width>` `X` `<excavation-depth>` `X` {`length`} `X` [**1**/ **27**])
Unit Cost = **{`excavation-unitcost`}**

restoration-overlay-cost
Description: *Overlay Pavement Restoration*

Unit Value: SY
Quantity = (*max*[([*lookup* {pavement-restoration} in **[pavement-restoration-width]**] + [**-1** X <min-pavement-width>]), **0**] X {length} X [**1**/ **9**])
Unit Cost = **{overlay-paving-unitcost}**

restoration-trench-cost
Description: *Complete Pavement Restoration* Unit Value: SY
Quantity = ([**1** + *if*({pavement-restoration}="**none**", **-1**, **0**)] X <min-pavement-width> X {length} X [**1**/ **9**])
Unit Cost = **{asphalt-paving-unitcost}**

row-cost
Description: *Land Acquisition* Unit Value: SF
Quantity = ({length} X <min-easment-width>)
Unit Cost = (*lookup* {row} in **[row-unitcost]**)

trench-safety-cost
Description: *Trench Safety* Unit Value: SF
Quantity = (**2** X <excavation-depth> X {length})
Unit Cost = (*if*[{trench-safety}="**standard**", **{trench-safety-unitcost}**, **0**] + *if*[{trench-safety}="**special**", **{special-shoring-unitcost}**, **0**])

Multiplier Definitions

mobilization-demobilization
Description: *Mobilization/Demobilization at 10 percent*
Multiplier = **1.1**

Year Definition

Year Calculation
Year = {`construction-year`}

Trenchless Conduit Template

Programatic name: **`trenchless-conduit-template`**
Not Public, hidden from user interface

Domain Definition

Description: *Trenchless Conduit Template*

Summary

Parameter List		
Name	**Description**	**Defined In**
`construction-year`	Construction Year	`trenchless-conduit-template`
`dewatering`	Dewatering	`trenchless-conduit-template`
`diameter`	Diameter	`trenchless-conduit-template`
`easment-length`	Tunnel Easement Length	`trenchless-conduit-template`
`easment-type`	Easement Type	`trenchless-conduit-template`
`length`	Length	`trenchless-conduit-`

		template
ls-excavation-depth	Launch Shaft Excavation Depth	trenchless-conduit-template
ls-surface-restoration	Launch Shaft Surface Restoration	trenchless-conduit-template
ls-utilities	Launch Shaft Existing Utilities	trenchless-conduit-template
rs-excavation-depth	Retrieval Shaft Excavation Depth	trenchless-conduit-template
rs-surface-restoration	Retrieval Shaft Surface Restoration	trenchless-conduit-template
rs-utilities	Retrieval Shaft Existing Utilities	trenchless-conduit-template

Cross Reference Lookup Table List	
Name	**Defined In**
surface-restoration-unitcost	trenchless-conduit-template
utilities-unitcost	trenchless-conduit-template

Constant Definitions

No Defined Constants

Parameter Definitions

Name	Description	Type	Default	Units
construction-year	Construction Year	year	2006	n/a
dewatering	Dewatering	enum	minimal	n/a
	Valid Values:	None = none Minimal = minimal Significant = significant		

`diameter`	Diameter	length	12	in
`easment-length`	Tunnel Easement Length	length	0	ft
`easment-type`	Easement Type	enum	none	n/a
	Valid Values:	None = none Residential = residential Industrial = industrial Office/Commercial = commercial		
`length`	Length	length	100	ft
`ls-excavation-depth`	Launch Shaft Excavation Depth	length	40	ft
		(Valid values range from 15.)		
`ls-surface-restoration`	Launch Shaft Surface Restoration	enum	hydroseed	n/a
	Valid Values:	None = none Hydroseed = hydroseed Pavement = pavement		
`ls-utilities`	Launch Shaft Existing Utilities	enum	average	n/a
	Valid Values:	None = none Average = average Complex = complex		
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	length	20	ft
		(Valid values range from 10.)		
`rs-surface-restoration`	Retrieval Shaft Surface Restoration	enum	hydroseed	n/a
	Valid Values:	None = none Hydroseed = hydroseed Pavement = pavement		
`rs-utilities`	Retrieval Shaft Existing Utilities	enum	average	n/a
	Valid Values:	None = none Average = average Complex = complex		

Cross Reference Table Definitions

surface-restoration-unitcost	
Value (enum)	**Result** ($/SY)
`none`	0
`hydroseed`	5
`pavement`	40

utilities-unitcost	
Value (enum)	**Result** ($/SF)
`none`	0
`average`	7
`complex`	12

Calculator Definition

Description: *Trenched Conduit Template 1.0*
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`ls-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`ls-easment-length`	Launch Shaft Easement Length	`trenchless-conduit-template`
`ls-easment-width`	Launch Shaft Easement Width	`trenchless-conduit-template`

`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-footprint`	Footprint	`trenchless-conduit-template`
`ls-length`	LS Length Placeholder	`trenchless-conduit-template`
`ls-volume`	Volume	`trenchless-conduit-template`
`ls-width`	LS Width Placeholder	`trenchless-conduit-template`
`outer-diameter`	Outer Diameter Placeholder	`trenchless-conduit-template`
`rs-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`rs-easment-length`	Retrieval Shaft Easement Length	`trenchless-conduit-template`
`rs-easment-width`	Retrieval Shaft Easement Width	`trenchless-conduit-template`
`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-footprint`	Footprint	`trenchless-conduit-template`
`rs-length`	RS Length Placeholder	`trenchless-conduit-template`
`rs-volume`	Volume	`trenchless-conduit-template`
`rs-width`	RS Width Placeholder	`trenchless-conduit-template`
`tunnel-spoil-loads`	Spoils Loads	`trenchless-conduit-template`
`tunnel-spoils`	Spoils Volume	`trenchless-conduit-template`

Line Item Cost List

Name	Description	Defined In
`ls-backfill-cost`	Launch Shaft Backfill	`trenchless-conduit-template`
`ls-easment-cost`	Launch Shaft Easement	`trenchless-conduit-template`
`ls-excavation-cost`	Launch Shaft Excavation	`trenchless-conduit-template`
`ls-shoring-cost`	Launch Shaft Shoring	`trenchless-conduit-template`
`ls-surface-restoration-cost`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilites-cost`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-backfill-cost`	Retrieval Shaft Backfill	`trenchless-conduit-template`
`rs-easment-cost`	Retrieval Shaft Easement	`trenchless-conduit-template`
`rs-excavation-cost`	Retrieval Shaft Excavation	`trenchless-conduit-template`
`rs-shoring-cost`	Retrieval Shaft Shoring	`trenchless-conduit-template`
`rs-surface-restoration-cost`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilites-cost`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`spoils-haul-cost`	Spoils Haul	`trenchless-conduit-template`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenchless-conduit-template`

Variable Definitions

ls-easment-footprint
Category: *Launch Shaft Geometry* Description: *Easement Footprint* Units: `SF`
Variable = (`<ls-easment-width> X <ls-easment-length>`)

ls-easment-length
Description: *Launch Shaft Easement Length* Units: `ft`
Variable = (**20** + *if*[`<ls-excavation-depth>` > **12**, **10**, **0**] + *if*[`<ls-excavation-depth>` > **20**, **20**, **0**] + `<ls-length>`)

ls-easment-width
Description: *Launch Shaft Easement Width* Units: `ft`
Variable = (**20** + *if*[`<ls-excavation-depth>` > **12**, **10**, **0**] + *if*[`<ls-excavation-depth>` > **20**, **20**, **0**] + `<ls-width>`)

ls-excavation-depth
Description: *Launch Shaft Excavation Depth* Units: `ft`
Variable = {`ls-excavation-depth`}

ls-footprint
Category: *Launch Shaft Geometry*

Description: *Footprint* Units: `SF`
Variable = (`<ls-width> X <ls-length>`)

ls-length
Description: *LS Length Placeholder* Units: `ft`
Variable = **1**

ls-volume
Category: *Launch Shaft Geometry* Description: *Volume* Units: `CY`
Variable = (`<ls-footprint> X {ls-excavation-depth} X` [**1**/ **27**])

ls-width
Description: *LS Width Placeholder* Units: `ft`
Variable = **1**

outer-diameter
Description: *Outer Diameter Placeholder* Units: `ft`
Variable = **0**

rs-easment-footprint
Category: *Retrieval Shaft Geometry* Description: *Easement Footprint* Units: `SF`
Variable = (`<rs-easment-width> X <rs-easment-length>`)

rs-easment-length
Description: *Retrieval Shaft Easement Length* Units: `ft`
Variable = (**20** + *if*[`<rs-excavation-depth>` > **12**, **10**, **0**] + *if*[`<rs-excavation-depth>` > **20**, **20**, **0**] + `<rs-length>`)

rs-easment-width
Description: *Retrieval Shaft Easement Width* Units: `ft`
Variable = (**20** + *if*[`<rs-excavation-depth>` > **12**, **10**, **0**] + *if*[`<rs-excavation-depth>` > **20**, **20**, **0**] + `<rs-width>`)

rs-excavation-depth
Description: *Retrieval Shaft Excavation Depth* Units: `ft`
Variable = {`rs-excavation-depth`}

rs-footprint
Category: *Retrieval Shaft Geometry* Description: *Footprint*

Units: `SF`
Variable = (`<rs-width> X <rs-length>`)

rs-length
Description: *RS Length Placeholder* Units: `ft`
Variable = **1**

rs-volume
Category: *Retrieval Shaft Geometry* Description: *Volume* Units: `CY`
Variable = (`<rs-footprint> X {rs-excavation-depth} X` [**1**/ **27**])

rs-width
Description: *RS Width Placeholder* Units: `ft`
Variable = **1**

tunnel-spoil-loads
Category: *Miscellaneous* Description: *Spoils Loads* Units: `loads`
Variable = *ceiling*([`<tunnel-spoils> X` (**1**/ **10**)])

tunnel-spoils
Category: *Tunnel Geometry* Description: *Spoils Volume* Units: `CY`
Variable = (`<outer-diameter>` `X` `<outer-diameter>` `X` `{length}` `X` **3.141592654** `X` [**1**/ **108**])

Line Item Cost Definitions

ls-backfill-cost
Description: *Launch Shaft Backfill* Unit Value: `CY`
Quantity = `<ls-volume>`
Unit Cost = **`{backfill-unitcost}`**

ls-easment-cost
Description: *Launch Shaft Easement* Unit Value: `SF`
Quantity = `<ls-footprint>`
Unit Cost = (*lookup* `{easment-type}` in **`[property-row-unitcost]`**)

ls-excavation-cost
Description: *Launch Shaft Excavation* Unit Value: `CY`
Quantity = `<ls-volume>`
Unit Cost = **`{shaft-excavation-unitcost}`**

ls-shoring-cost
Description: *Launch Shaft Shoring* Unit Value: `SF`
Quantity = (**2** `X` {`ls-excavation-depth`} `X` [`<ls-width> + <ls-length>`])
Unit Cost = (**10** `+` [**1.85** `X` {`ls-excavation-depth`}])

ls-surface-restoration-cost
Description: *Launch Shaft Surface Restoration* Unit Value: `SY`
Quantity = (`<ls-footprint>` `X` [**1**/ **9**])
Unit Cost = (*lookup* {`ls-surface-restoration`} in **[`surface-restoration-unitcost`]**)

ls-utilites-cost
Description: *Launch Shaft Existing Utilities* Unit Value: `SF`
Quantity = `<ls-footprint>`
Unit Cost = (*lookup* {`ls-utilities`} in **[`utilities-unitcost`]**)

rs-backfill-cost
Description: *Retrieval Shaft Backfill* Unit Value: `CY`
Quantity = `<rs-volume>`
Unit Cost = **{`backfill-unitcost`}**

rs-easment-cost
Description: *Retrieval Shaft Easement* Unit Value: `SF`
Quantity = `<rs-footprint>`
Unit Cost = (*lookup* {`easment-type`} in **[`property-row-unitcost`]**)

rs-excavation-cost
Description: *Retrieval Shaft Excavation* Unit Value: `CY`
Quantity = `<rs-volume>`
Unit Cost = **{`shaft-excavation-unitcost`}**

rs-shoring-cost
Description: *Retrieval Shaft Shoring* Unit Value: `SF`
Quantity = (**2** `X` {`rs-excavation-depth`} `X` [`<rs-width> + <rs-length>`])
Unit Cost = (**10** `+` [**1.85** `X` {`rs-excavation-depth`}])

rs-surface-restoration-cost
Description: *Retrieval Shaft Surface Restoration* Unit Value: `SY`
Quantity = (`<rs-footprint>` `X` [**1**/ **9**])
Unit Cost = (*lookup* {`rs-surface-restoration`} in **[`surface-restoration-unitcost`]**)

rs-utilites-cost
Description: *Retrieval Shaft Existing Utilities* Unit Value: `SF`
Quantity = `<rs-footprint>`
Unit Cost = (*lookup* {`rs-utilities`} in **[`utilities-unitcost`]**)

spoils-haul-cost
Description: *Spoils Haul* Unit Value: `CY`
Quantity = `<tunnel-spoils>`
Unit Cost = **{`spoil-haul-unitcost`}**

Multiplier Definitions

mobilization-demobilization
Description: *Mobilization/Demobilization at 10 percent*
Multiplier = **1.1**

Year Definition

Year Calculation
Year = {`construction-year`}

Tunnel

Programatic name: **`tunnel`**
Specialized User Interface: `tabula.model.trenchless.TunnelInfoFrame`
Customized Icon Path: `tabula/model/tunnel/tree_icon.gif`

Domain Definition

Description: *Tunnel*
Inherits from *Trenchless Conduit Template* `(trenchless-conduit-template)`

Summary

Constant List	
Name	**Defined In**
`backfill-unitcost`	`tunnel`
`shaft-excavation-unitcost`	`tunnel`
`spoil-haul-unitcost`	`tunnel`

Parameter List		
Name	**Description**	**Defined In**
`construction-year`	Construction Year	`trenchless-conduit-template`
`dewatering`	Dewatering	`trenchless-conduit-template`
`diameter`	Inside Diameter	`tunnel`
`easment-length`	Tunnel Easement Length	`trenchless-conduit-template`
`easment-type`	Easement Type	`trenchless-conduit-template`
`length`	Length	`trenchless-conduit-template`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-footprint-type`	Launch Shaft Footprint	`tunnel`
`ls-surface-restoration`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilities`	Launch Shaft Existing Utilities	`trenchless-conduit-template`

`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-footprint-type`	Retrieval Shaft Footprint	`tunnel`
`rs-surface-restoration`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilities`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`

Cross Reference Lookup Table List	
Name	**Defined In**
`outside-diameter`	`tunnel`
`property-row-unitcost`	`tunnel`
`surface-restoration-unitcost`	`trenchless-conduit-template`
`tbm-procurment`	`tunnel`
`tunnel-boring-unitcost`	`tunnel`
`tunneling-row-unitcost`	`tunnel`
`utilities-unitcost`	`trenchless-conduit-template`

Constant Definitions

Name	Type	Value	Units
`backfill-unitcost`	cost	18	$/CY
`shaft-excavation-unitcost`	cost	15	$/CY
`spoil-haul-unitcost`	cost	12	$/CY

Parameter Definitions

Name	Description	Type	Default	Units

`diameter`	Inside Diameter	enum	8	n/a
			Valid Values:	8 ft. = 8 9 ft. = 9 10 ft. = 10 11 ft. = 11 12 ft. = 12 13 ft. = 13 14 ft. = 14 15 ft. = 15 16 ft. = 16 18 ft. = 18
`ls-footprint-type`	Launch Shaft Footprint	enum	standard	n/a
			Valid Values:	Standard = standard Oversized = oversized
`rs-footprint-type`	Retrieval Shaft Footprint	enum	standard	n/a
			Valid Values:	Standard = standard Oversized = oversized

Parameters inherited from **`trenchless-conduit-template`**.
`construction-year, dewatering, easment-length, easment-type, length, ls-excavation-depth, ls-surface-restoration, ls-utilities, rs-excavation-depth, rs-surface-restoration, rs-utilities`

Cross Reference Table Definitions

outside-diameter	
Value (ft)	**Result** (ft)
8	9.25

9	10.25
10	11.33
11	12.33
12	13.33
13	14.33
14	15.5
15	16.5
16	17.5
18	19.5

property-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	58
industrial	23
commercial	37

tbm-procurment	
Value (ft)	**Result** ($)
8	2000000
9	2500000
10	2750000
11	3000000
12	3500000
13	3750000

14	4000000
15	4500000
16	5000000
18	5500000

tunnel-boring-unitcost	
Value (ft)	**Result** ($/lf)
8	2200
9	2200
10	2200
11	2200
12	2400
13	2600
14	2700
15	2900
16	3200
18	3400

tunneling-row-unitcost	
Value (enum)	**Result** ($/sf)
none	0
residential	17
industrial	7
commercial	11

Cross Reference Tables inherited from **`trenchless-conduit-template`**.
`surface-restoration-unitcost, utilities-unitcost`

Calculator Definition

Description: *Tunnel 1.0*
Inherits from *Trenched Conduit Template 1.0* `(trenchless-conduit-template)`
Basis Year: **`2005`**

Summary

Variable List		
Name	**Description**	**Defined In**
`ls-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`ls-easment-length`	Launch Shaft Easement Length	`trenchless-conduit-template`
`ls-easment-width`	Launch Shaft Easement Width	`trenchless-conduit-template`
`ls-excavation-depth`	Launch Shaft Excavation Depth	`trenchless-conduit-template`
`ls-footprint`	Footprint	`trenchless-conduit-template`
`ls-length`	Length	`tunnel`
`ls-volume`	Volume	`trenchless-conduit-template`
`ls-width`	Width	`tunnel`
`outer-diameter`	Outer Diameter	`tunnel`
`rs-easment-footprint`	Easement Footprint	`trenchless-conduit-template`
`rs-easment-length`	Retrieval Shaft Easement Length	`trenchless-conduit-template`
`rs-easment-width`	Retrieval Shaft Easement Width	`trenchless-conduit-template`

`rs-excavation-depth`	Retrieval Shaft Excavation Depth	`trenchless-conduit-template`
`rs-footprint`	Footprint	`trenchless-conduit-template`
`rs-length`	Length	`tunnel`
`rs-volume`	Volume	`trenchless-conduit-template`
`rs-width`	Width	`tunnel`
`tunnel-length`	Tunnel Length	`tunnel`
`tunnel-spoil-loads`	Spoils Loads	`trenchless-conduit-template`
`tunnel-spoils`	Spoils Volume	`trenchless-conduit-template`

Line Item Cost List		
Name	**Description**	**Defined In**
`dewatering-cost`	Tunnel Dewatering	`tunnel`
`ls-backfill-cost`	Launch Shaft Backfill	`trenchless-conduit-template`
`ls-easment-cost`	Launch Shaft Easement	`trenchless-conduit-template`
`ls-excavation-cost`	Launch Shaft Excavation	`trenchless-conduit-template`
`ls-shoring-cost`	Launch Shaft Shoring	`trenchless-conduit-template`
`ls-surface-restoration-cost`	Launch Shaft Surface Restoration	`trenchless-conduit-template`
`ls-utilites-cost`	Launch Shaft Existing Utilities	`trenchless-conduit-template`
`rs-backfill-cost`	Retrieval Shaft Backfill	`trenchless-conduit-template`
`rs-easment-cost`	Retrieval Shaft Easement	`trenchless-conduit-`

		`template`
`rs-excavation-cost`	Retrieval Shaft Excavation	`trenchless-conduit-template`
`rs-shoring-cost`	Retrieval Shaft Shoring	`trenchless-conduit-template`
`rs-surface-restoration-cost`	Retrieval Shaft Surface Restoration	`trenchless-conduit-template`
`rs-utilites-cost`	Retrieval Shaft Existing Utilities	`trenchless-conduit-template`
`spoils-haul-cost`	Spoils Haul	`trenchless-conduit-template`
`tmb-procurment-cost`	TBM Procurment	`tunnel`
`tunnel-boring-cost`	Tunnel Boring	`tunnel`
`tunnel-easment-cost`	Tunnel Easement	`tunnel`

Subtotal Cost Multiplier List		
Name	**Description**	**Defined In**
`mobilization-demobilization`	Mobilization/Demobilization at 10 percent	`trenchless-conduit-template`

Variable Definitions

ls-length
Category: *Launch Shaft Geometry* Description: *Length* Units: `ft`
Variable = *ceiling*([`<outer-diameter> X` (*if*[{`ls-footprint-type`}=**"standard"**, **9.5**, **0**] + *if*[{`ls-footprint-type`}=**"oversized"**, **12**, **0**])])

ls-width
Category: *Launch Shaft Geometry* Description: *Width* Units: `ft`
Variable = *ceiling*([`<outer-diameter> X` (*if*[{`ls-footprint-type`}="**standard**", **3**, **0**] `+` *if*[{`ls-footprint-type`}="**oversized**", **5**, **0**])])

outer-diameter
Category: *Tunnel Geometry* Description: *Outer Diameter* Units: `ft`
Variable = (*lookup* {`diameter`} in **[`outside-diameter`]**)

rs-length
Category: *Retrieval Shaft Geometry* Description: *Length* Units: `ft`
Variable = *ceiling*([`<outer-diameter> X` (*if*[{`rs-footprint-type`}="**standard**", **3.5**, **0**] `+` *if*[{`rs-footprint-type`}="**oversized**", **6**, **0**])])

rs-width
Category: *Retrieval Shaft Geometry* Description: *Width* Units: `ft`
Variable = *ceiling*([`<outer-diameter> X` (*if*[{`rs-footprint-type`}="**standard**", **2.5**, **0**] `+` *if*[{`rs-footprint-type`}="**oversized**", **4**, **0**])])

tunnel-length
Description: *Tunnel Length* Units: `ft`
Variable = {`length`}

Variables inherited from **`trenchless-conduit-template`**.
`ls-easment-footprint, ls-easment-length, ls-easment-width, ls-excavation-depth, ls-footprint, ls-volume, rs-easment-footprint, rs-easment-length, rs-easment-width, rs-excavation-depth, rs-footprint, rs-volume, tunnel-spoil-loads, tunnel-spoils`

Line Item Cost Definitions

dewatering-cost
Description: *Tunnel Dewatering* Unit Value: `LS`
Quantity = **1**
Unit Cost = (*if*[{`dewatering`}=**"minimal"**, (**50000** + *if*[`<tunnel-length>` > **1000**, **5000**, **0**] + *if*[`<tunnel-length>` > **5000**, **5000**, **0**] + *if*[`<tunnel-length>` > **10000**, **10000**, **0**]), **0**] + *if*[{`dewatering`}=**"significant"**, (**70000** + *if*[`<tunnel-length>` > **1000**, **10000**, **0**] + *if*[`<tunnel-length>` > **5000**, 2**5000**, **0**] + *if*[`<tunnel-length>` > **10000**, **15000**, **0**]), **0**])

tmb-procurment-cost
Description: *TBM Procurment* Unit Value: `LS`
Quantity = **1**
Unit Cost = (*lookup* {`diameter`} in **[`tbm-procurment`]**)

tunnel-boring-cost
Description: *Tunnel Boring* Unit Value: `ft`
Quantity = {`length`}
Unit Cost = (*lookup* {`diameter`} in **[`tunnel-boring-unitcost`]**)

tunnel-easment-cost
Description: *Tunnel Easement* Unit Value: `SF`
Quantity = ({`easment-length`} `X` [`<outer-diameter> +` **20**])
Unit Cost = (*lookup* {`easment-type`} in **[`tunneling-row-unitcost`]**)

Line item costs inherited from **`trenchless-conduit-template`**.
`ls-backfill-cost, ls-easment-cost, ls-excavation-cost, ls-shoring-cost, ls-surface-restoration-cost, ls-utilites-cost, rs-backfill-cost, rs-easment-cost, rs-excavation-cost, rs-shoring-cost, rs-surface-restoration-cost, rs-utilites-cost, spoils-haul-cost`

Multiplier Definitions

No Defined Multipliers

Multipliers inherited from **`trenchless-conduit-template`**.
`mobilization-demobilization`

Year Definition

Year Calculation

Year = `{construction-year}`