Distribution Tool 1.0 Component Specification

1.Design

For each component design TopCoder provides a distribution which includes directory structure, build files, requirements documents etc. Creating these distributions manually for each component is very laborious and so a special tool was developed to automate the process. Originally the tool was a command line utility developed using C++, and this component replaces this tool with new one written in Java. The new tool provides an API for the application (cockpit) to create the distributions automatically.

DistributionTool is the main class of this component. It is configured using Configuration Persistence component. A configuration file contains paths to distribution script files and lists of required/default parameters for these scripts.

DistributionTool parses all script files during the initialization using a pluggable DistributionScriptParser instance.

To create a distribution the user needs to specify the distribution type (each distribution type is associated with one script) and a map of input string parameters. There are 3 types of parameters:

- Required – must be always specified by the user in the map.
- Optional with default value – when not specified by the user, some configured default value is used instead.
- Optional – is not mentioned in the configuration and can be omitted in the input parameters map, but can be handled by the script itself using “ifdef” and “ifndef” keywords.

The parsed distribution script consists of a sequence of distribution script commands. Each command is represented with a DistributionScriptCommand instance.

Currently the following DistributionScriptCommand implementations are defined in this component: DefineVariableCommand, UndefineVariableCommand, CreateFolderCommand, CopyFileCommand, CopyFileTemplateCommand, ExecuteProcessCommand and ConvertToPDFCommand.

1.1Design Patterns

Strategy pattern – DistributionTool uses implementations of DistributionScriptCommand and DistributionScriptParser interfaces as strategies; BaseDistributionScriptCommand uses pluggable CommandExecutionCondition implementation instances as strategies.

Template method pattern – BaseDistributionScriptCommand#execute() is a template method that uses an abstract executeCommand() method.

1.2Industry Standards

None

1.3Required Algorithms

1.3.1Script file format

This section describes the format of the distribution script file used by DistributionScriptParserImpl.

This component uses the old script file format (used in C++ command line utility) with some new commands and extensions.

Empty lines (after trimming) in the script files are ignored.

Lines that start with ‘#’ (after trimming) are considered to be comments and are ignored too.

All other lines in the script file are interpreted as command definitions; one command per line. Such lines must have the following format:

```
[ ifdef(variable_name) | ifndef(variable_name) ]* command_name : command_body
```

All whitespaces between separators are ignored. Command names, “ifdef” and “ifndef” keywords are case insensitive. “*” means that zero, one or multiple ifdef(...) and/or ifndef(...) conditions can be specified before command name, separated with whitespace(s). If multiple conditions are specified for a single command, they are ANDed. E.g. the following script line means that variable with name “var3” must be set to “ABC” only if a variable with name “var1” is defined AND a variable with name “var2” is not defined:

```
ifdef(var1) ifndef(var2) define : var3 = ABC
```

Command names supported by this component are fixed. Command body holds parameters required for performing specific operation. Thus each command has its own meaning of “command_body” field.

All fields of command_body can contain field variables in format “%{variable_name}”. E.g.:

```
create_folder: %{comp_dir}/conf
```

In this case the component will replace “%{comp_dir}” substring with the value of “comp_dir” variable from the current execution context (values of these variables are not saved between executions of the same script).

The following 3 variables are always defined by DistributionTool before executing each script (note that these variables can still be undefined with “undefine” command inside the script):

- distribution_type – the distribution type associated with this script; mapping between distribution types and script files is defined in the configuration of DistributionTool;
- current_year – the current year in format “yyyy”;
- script_dir – the full path of the folder in which the script file is located.

Additionally each input parameter passed by the user to createDistribution() method inside the “parameters” map becomes a namesake variable that can be accessed from the script.

To support old distribution source files (that are located in “files” folders) used with C++ command line utility, the following input parameters are processed in special way:

- version – it’s assumed that value of this parameter can have “a”, “a.b”, “a.b.c” or “a.b.c.d” format (dots are used to separate major, minor, micro version and build numbers). If this parameter is specified by the user, DistributionTool additionally defines the following variables:
  - version.major – equals to the first dot-separated part of version (i.e. “a”);
  - version.minor – equals to the second dot-separated part of version (i.e. “b”);
  - version.micro – equals to the third dot-separated part of version (i.e. “c”);
  - version.build – equals to the forth dot-separated part of version (i.e. “d”);
- Component Name – if specified DistributionTool additionally defines the following variables:
  - component name – value of “Component Name” parameter converted to lower case;
  - component_name – value of “Component Name” parameter converted to lower case, and with space character replaced with underscore;
  - Component_Name – value of “Component Name” parameter with space character replaced with underscore;
- package.name – if specified DistributionTool additionally defines the following variables:
  - package/name – value of “package.name” parameter with dot replaced with “/” character;

- package\name – value of “package.name” parameter with dot replaced with “\” character.

The table below describes all command names supported by this component (note that command names are case insensitive):

command_name	Description of command and command_body
define	Defines a variable. If already defined, modifies the value of the variable. If variable is defined “ifdef(...)” condition is met, and “ifndef(...)” condition is not met. Format of command_body: `variable_name = variable_value`
undefine	Undefines a variable. Does nothing if variable is not defined. If variable is not defined “ifndef(...)” condition is met, and “ifdef(...)” condition is not met. Format of command_body: `variable_name`
create_folder	Creates a folder. All parent folders are created recursively if required. Format of command_body: `folder_path` folder_path must be a full path of the folder to be created.
copy_file	Copies a file from one location to another on file system. Format of command_body: `source_path -> destination_path` Both fields must be full file paths. destination_path can contain "{FILENAME}" and "{EXT}" keywords that are replaced with the source file name (without extension) and the source file extension accordingly. Note that variable fields are processed before “{FILENAME}” and “{EXT}”.
file_template	Copies a text file from one location to another on the file system and replaces all variable fields in format "%{variable_name}" in the destination file with variable values from the execution context. Format of command_body: `source_path -> destination_path` Both fields must be full file paths. destination_path can contain "{FILENAME}" and "{EXT}" keywords that are replaced with the source file name (without extension) and the source file extension accordingly. Note that variable fields are processed before “{FILENAME}” and “{EXT}”.
execute	Executes an external process using the specified command line and waits until this process is terminated. Format of command_body: `command_line :: working_path` working_path must be a full path of the directory that is used as a working folder for a created process.

convert_to_pdf	Converts the given input file to PDF format and copies it to the specified location on the file system. Format of command_body: `source_path -> destination_path` Can convert files of DOC, RTF, HTML, TXT, PS and some other formats. Simply copies a source file if it has PDF extension.

1.3.2Parsing script and executing script commands

These algorithms are very straightforward. You can find their description in implementation notes for DistributionScriptParserImpl#parseCommands(stream, script, log) and XxxCommand#executeCommand() methods provided in TCUML file.

1.4Component Class Overview

BaseDistributionScriptCommand [abstract]

This is a base class for all DistributionScriptCommand implementations provided in this component. It provides a common logic for all commands: supports multiple CommandExecutionCondition instances being associated with the command, holds a Log instance and provides methods for replacing variable fields in the given string. This class logs all unknown variable names at WARN level using the provided Logging Wrapper logger. Subclasses should log information about performed operations at INFO level for consistency; they can access the logger using getLog() method. All subclasses must implement executeCommand() method that is called when command execution conditions are met only.

CommandExecutionCondition [interface]

This interface represents a command execution condition. This condition must be met in case if the associated script command should be executed. This interface defines only a single method for checking whether the condition is met.

ConvertToPDFCommand

This class is an implementation of DistributionScriptCommand that converts the given input file to PDF format and copies it to the specified location on the filesystem. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component. This class uses PDF File Conversion component to convert files to PDF format, thus it supports source files of DOC, RTF, HTML, TXT, PS and some other formats. When the source file has PDF extension, the file is simply copied without any conversion performed.

CopyFileCommand

This class is an implementation of DistributionScriptCommand that copies a file from one location to another on the filesystem. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component. This command doesn't change the content of the file, but can rename the file if required. Destination path can contain "FILENAME" and "EXT" keywords that are replaced with the source file name (without extension) and the source file extension accordingly.

CopyFileTemplateCommand

This class is an implementation of DistributionScriptCommand that copies a text file from one location to another on the filesystem and replaces all variable fields in format "%variable_name" in the destination file with variable values from the execution context. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component. This command can rename the file if required.

Destination path can contain "FILENAME" and "EXT" keywords that are replaced with the source file name (without extension) and the source file extension accordingly.

CreateFolderCommand

This class is an implementation of DistributionScriptCommand that creates a folder on the filesystem. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component. Note that this command creates all parent folders recursively if required.

DefineVariableCommand

This class is an implementation of DistributionScriptCommand that defines a variable in the current script execution context. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component.

DistributionScript

This class is a container for distribution script information. It holds the list of commands in the script, the lists of required and default parameters and the path of the folder where the script file is located. This class provides getters and setter for all script parameters stored in private fields. Shallow copies of collections are used in setters and getters.

DistributionScriptCommand [interface]

This interface represents a distribution script command. Each command implementation must hold parameters and logic required for performing specific operation used when creating a distribution. Sequence of commands form a distribution script. Implementations can use the given DistributionScriptExecutionContext instance to access the current context variables.

DistributionScriptExecutionContext

This class is a container for distribution script execution context information. Currently it just holds a list of defined context variables. Variable names and values are strings, variable values cannot be null. This class is used by DistributionTool and DistributionScriptCommand to share information between command executions.

DistributionScriptParser [interface]

This interface represents a distribution script parser. It defines a single method for parsing the list of distribution script commands from the given input stream and setting them to the given DistributionScript instance.

DistributionScriptParserImpl

This class is an implementation of DistributionScriptParser that can read distribution scripts in format described in the section 1.3.1 and supports all commands defined in the com.topcoder.util.distribution.commands package of this component. Currently the following DistributionScriptCommand implementations are supported by this class: DefineVariableCommand, UndefineVariableCommand, CreateFolderCommand, CopyFileCommand, CopyFileTemplateCommand, ExecuteProcessCommand and ConvertToPDFCommand.

DistributionTool

This is the main class of this component. It can be used for creating distributions of different types. For each supported distribution type this class holds an associated DistributionScript instance. This script is executed when creating a distribution of this type. DistributionTool can be configured using Configuration API and Configuration Persistence components. In this case distribution scripts are read from text files and parsed using DistributionScriptParser instance. DistributionScriptParser instance to be used for this is created with Object Factory. Most of input parameters for distribution script are provided by the user in a string map. But this class can additionally use default parameter values specified in the configuration. All input parameters are used as context variables when executing script commands. Additionally "version.major", "version.minor", "version.micro" and "version.build" variables can be constructed from "version" parameter; "component name", "component_name" and "Component_Name" - from "Component Name" parameter; "package/name" and "package\name" - from "package.name" parameter. Also DistributionTool always defines the following variables: "distribution_type", "current_year" and "script_dir".

ExecuteProcessCommand

This class is an implementation of DistributionScriptCommand that executes an external process using the specified command line and waits until this process is terminated. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component. This command uses a specified working directory when starting a process.

UndefineVariableCommand

This class is an implementation of DistributionScriptCommand that undefines a variable in the current script execution context. It extends BaseDistributionScriptCommand that provides common functionality for all commands defined in this component.

VariableDefinedCondition

This class is an implementation of CommandExecutionCondition that checks whether variable with specific name is defined in the current execution context.

VariableNotDefinedCondition

This class is an implementation of CommandExecutionCondition that checks whether variable with specific name is not defined in the current execution context.

1.5Component Exception Definitions

DistributionScriptCommandExecutionException

This exception is thrown by DistributionTool and implementations of DistributionScriptCommand when some error occurs while executing one of script commands. This exception can be used as a base for other command-specific exceptions.

DistributionScriptParsingException

This exception is thrown by implementations of DistributionScriptParser when some error occurs while parsing a distribution script obtained from the given input stream.

DistributionToolConfigurationException

This runtime exception is thrown by DistributionTool when some error occurs while reading the configuration of this class (e.g. when required parameter is missing or has invalid format) or initializing the class with this configuration.

DistributionToolException

This exception is a base class for all other non-runtime custom exceptions defined in this component. It is never thrown directly, subclasses are used instead.

InputFileNotFoundException

This exception is thrown by CopyFileCommand, CopyFileTemplateCommand and ConvertToPDFCommand when a source file to be used by the command cannot be found.

MissingInputParameterException

This exception is thrown by DistributionTool when any of input parameters required for distribution script execution is missing in the parameters map provided by the user.

PDFConversionException

This exception is thrown by ConvertToPDFCommand when some error occurs while converting a source file to PDF format.

ProcessExecutionException

This exception is thrown by ExecuteProcessCommand when some error occurs while starting an external process.

UnknownDistributionTypeException

This exception is thrown by DistributionTool when the distribution type specified by the user is unknown (is not supported by this DistributionTool instance).

1.6Thread Safety

This component is thread safe.

DistributionTool is immutable and thread safe. It uses DistributionScript instances in thread safe manner (and assumes that the caller uses them in thread safe manner too when these instances are provided by the caller). DistributionScriptCommand instances used by this class are thread safe.

DistributionScriptExecutionContext has mutable variables collection, thus it's not thread safe. But in this component it's always accessed from a single thread only, thus it's used in thread safe manner.

DistributionScript is mutable and not thread safe. To use this class in thread safe manner it first must be initialized via setters from a single thread, and next getters can be used from multiple threads at a time.

Implementations of DistributionScriptCommand and CommandExecutionCondition must be thread safe assuming that DistributionScriptExecutionContext instance is used by the caller in thread safe manner.

BaseDistributionScriptCommand, DefineVariableCommand, UndefineVariableCommand, CreateFolderCommand, CopyFileCommand, CopyFileTemplateCommand, ExecuteProcessCommand and ConvertToPDFCommand are immutable and thread safe when DistributionScriptExecutionContext instance is used by the caller in thread safe manner. They use thread safe Log instance.

VariableDefinedCondition and VariableNotDefinedCondition are immutable and thread safe when DistributionScriptExecutionContext instance is used by the caller in thread safe manner.

Implementations of DistributionScriptParser must be thread safe assuming that the given InputStream and DistributionScript instances are used by the caller in thread safe manner.

DistributionScriptParserImpl is immutable and thread safe when the given InputStream and DistributionScript instances are used by the caller in thread safe manner.

2.Environment Requirements

2.1Environment

Development language: Java 1.5
Compile target: Java 1.5, Java 1.6
QA Environment: Java 1.5, RedHat Linux 4, Windows 2000, Windows 2003

2.2TopCoder Software Components

Base Exception 2.0 – is used by custom exception defined in this component.

Configuration API 1.0 – is used for initializing classes from this component.

Configuration Persistence 1.0.2 – is used for reading configuration from file.

Logging Wrapper 2.0 – is used for logging warning and debug information.

Object Factory 2.1.2 – is used for creating pluggable object instances.

Object Factory Configuration API Plugin 1.0 – allows to configure Object Factory component with Configuration API object.

PDF File Conversion 1.0 – is used for converting files to PDF format.

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3Third Party Components

None

3.Installation and Configuration

3.1Package Name

com.topcoder.util.distribution
com.topcoder.util.distribution.commands
com.topcoder.util.distribution.commands.conditions
com.topcoder.util.distribution.parsers

3.2Configuration Parameters

3.2.1Configuration of DistributionTool

The following table describes the structure of ConfigurationObject passed to the constructor of DistributionTool class (angle brackets are used for identifying child configuration objects).

Parameter	**Description**	**Values**
logger_name	The logger name passed to LogManager when creating a Log instance passed to created commands. When not provided, logging is not performed by commands.	String. Not empty. Optional.
<object_factory_config>	This section contains configuration of Object Factory used by this class for creating DistributionScriptParser instance.	ConfigurationObject. Required.
script_parser_key	The Object Factory key that is used for creating an instance of DistributionScriptParser used by this class for parsing script files.	String. Not empty. Required.
<scripts>	The configuration section that contain information about all distribution scripts used by this tool.	ConfigurationObject. Required.
<scripts>.<script_XXX> where XXX is any substring	The configuration for a script used to create distributions of specific type.	ConfigurationObject. Multiple. At least one is required.
<scripts>.<script_XXX>. distribution_type	The distribution type associated with the script. Must be unique within the current <scripts> configuration.	String. Not empty. Required.

<scripts>.<script_XXX>.script_path	The path of the script file on the file system.	String. Not empty. Required.
<scripts>.<script_XXX>.<param_YYY> where YYY is any substring	The configuration for a single parameter that is passed to the script. Each input parameter is used for defining a namesake context variable. This section must represent only parameters for which variables MUST be defined before executing the script. If parameter is optional and doesn't have a default value it should not be mentioned in the configuration (the script can handle such parameters using "ifdef" and "ifndef" keywords).	ConfigurationObject. Multiple. Optional.
<scripts>.<script_XXX>.<param_YYY>.name	The name of the parameter.	String. Not empty. Required.
<scripts>.<script_XXX>.<param_YYY>.defaultValue	The default value of the parameter. If not specified, the parameter is required (i.e. "not optional" and must be always specified by the user).	String. Optional.

3.3Dependencies Configuration

None

4.Usage Notes

4.1Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2Required steps to use the component

Please see the demo.

4.3Demo

4.3.1Sample DistributionTool configuration

```
<?xml version="1.0"?>
<CMConfig>
  <Config name="com.topcoder.util.distribution.DistributionTool">
    <Property name="logger_name">
      <Value>my_logger</Value>
    </Property>
    <Property name="object_factory_config">
      <!-- Put Object Factory configuration here -->
    </Property>
    <Property name="script_parser_key">
      <Value>DistributionScriptParserImpl</Value>
    </Property>
    <Property name="scripts">
      <Property name="script_dotnet">
        <Property name="distribution_type">
```

```xml
          <Value>dotnet</Value>
        </Property>
        <Property name="script_path">
          <Value>scripts/dotnet/script.txt</Value>
        </Property>
        <Property name="param_1">
          <Property name="name">
            <Value>output_dir</Value>
          </Property>
        </Property>
        <Property name="param_2">
          <Property name="name">
            <Value>version</Value>
          </Property>
          <Property name="defaultValue">
            <Value>1.0.0.1</Value>
          </Property>
        </Property>
        <Property name="param_3">
          <Property name="name">
            <Value>Component Name</Value>
          </Property>
        </Property>
        <Property name="param_4">
          <Property name="name">
            <Value>package.name</Value>
          </Property>
        </Property>
        <Property name="param_5">
          <Property name="name">
            <Value>component_description</Value>
          </Property>
          <Property name="defaultValue">
            <Value></Value>
          </Property>
        </Property>
        <Property name="param_6">
          <Property name="name">
            <Value>rs</Value>
          </Property>
        </Property>
      </Property>
      <Property name="script_java">
        <Property name="distribution_type">
          <Value>java</Value>
        </Property>
        <Property name="script_path">
          <Value>scripts/java/script.txt</Value>
        </Property>
        <!-- Put here the same params config as for script_dotnet -->
      </Property>
      <Property name="script_flex">
        <Property name="distribution_type">
          <Value>flex</Value>
        </Property>
        <Property name="script_path">
          <Value>scripts/flex/script.txt</Value>
        </Property>
        <!-- Put here the same params config as for script_dotnet -->
      </Property>
      <Property name="script_js">
        <Property name="distribution_type">
          <Value>js</Value>
        </Property>
        <Property name="script_path">
          <Value>scripts/js/script.txt</Value>
        </Property>
        <!-- Put here the same params config as for script_dotnet -->
      </Property>
    </Property>
  </Config>
</CMConfig>
```

4.3.2Sample distribution scripts

You can find sample .NET, Flex, Java and JavaScript design distribution generation scripts in the "scripts" folder provided together with this document.

These scripts are almost identical to original script files used with C++ command line tool. Only the following modifications were made:

- Some folders creation commands were removed (assuming that parent folders are created recursively by this component).
- RS is now processed with "convert_to_pdf" command instead of "copy_file".
- 3 commands were added to include up to 3 additional documents into the distribution. Paths to these documents can be specified with "additional_doc1", "additional_doc2" and "additional_doc3" parameters. Original file names for these documents are preserved in the created distribution.
- "execute" command was added to execute Ant or MSBuild process to pack distribution into JAR or ZIP file.

Note that all files inside of "files" folders were not modified.

4.3.3API usage

```
// Create an instance of DistributionTool using configuration object
ConfigurationObject config = ...
DistributionTool distributionTool = new DistributionTool(config);

// Create an instance of DistributionTool using custom config file and namespace
distributionTool = new DistributionTool("my_config.properties", "my_namespace");

// Create an instance of DistributionTool using custom scripts mapping
Map<String, DistributionScript> scripts = new HashMap<String, DistributionScript>();
DistributionScript javaScript = ...
DistributionScript dotnetScript = ...
scripts.put("java", javaScript);
scripts.put("dotnet", dotnetScript);
distributionTool = new DistributionTool(scripts);

// Create an instance of DistributionTool using the default configuration file and namespace
distributionTool = new DistributionTool();

// Create Java design distribution for Test Component 1.1
Map<String, String> parameters = HashMap<String, String>();
parameters.add("output_dir", "c:/tc_dist");
parameters.add(DistributionTool.VERSION_PARAM_NAME, "1.1");
parameters.add(DistributionTool.COMPONENT_NAME_PARAM_NAME, "Test Component");
parameters.add(DistributionTool.PACKAGE_NAME_PARAM_NAME, "com.topcoder.test");
parameters.add("rs", "c:/test/testrs.doc");
parameters.add("additional_doc1", "c:/test/erd.png");
distributionTool.createDistribution("java", parameters);
```

4.3.4Sample output

Assuming that configuration provided in the section 4.3.1 and scripts mentioned in the section 4.3.2 are used, and distribution is created using the code provided in the section 4.3.3, the distribution output folder must have the following structure:

```
c:/tc_dist/Test Component/
                          conf/
                               putYourConfigFilesHere.txt
                          docs/
                               javadocs/
                                        stylesheet.css
                               Test_Component_Requirements_Specification.pdf
                               erd.png
                          test_files/
                                     putYourTestFilesHere.txt
                          src/
                              java/
```

```
                  main/
                       com/
                           topcoder/
                                    test/
                                         putYourSourceFilesHere.txt
                  tests/
                        com/
                            topcoder/
                                     test/
                                          accuracytests/
                                                        AccuracyTests.java
                                          failuretests/
                                                       FailureTests.java
                                          stresstests/
                                                      StressTests.java
                                          AllTests.java
                                          UnitTests.java
         build.version
         build.xml
         build-dependencies.xml
         build-dist.xml
         readme
         test_component_1.1.0_design_dist.jar
```

Below is the content of the created build.version file:

```
# Property file defining the component's information.
component.name=Test Component
component.distfilename=test_component
component.package=com.topcoder.test
component.packagedir=com/topcoder/test
component.version.major=1
component.version.minor=1
component.version.micro=0
component.version.build=1

# Defines the Main-Class property for the manifest file (e.g. com.topcoder.utility.SomeClass)
# This property is only used for components that can be run from command line.
# Leave the property value empty if not applicable.
component.mainclass=
```

5.Future Enhancements

Other command implementations can be provided.