Legacy Transform Device Web Services 1.0 Component Specification

1. Design

The client is operating a legacy version of the system that is in-process of being replaced. The legacy system includes custom hardware devices, network software, and Windows desktop applications. This component will create a collection of Web Service interfaces into the new system for the legacy desktop application. Each web service will need to transform the client request from the legacy XSD schema (attached) into the internal domain model to execute the operation.

1.1 Design Patterns

The DeviceService is a **façade** to the existing device repository system, providing a single point of service.

1.2 Industry Standards

WCF, XML

1.3 Required Algorithms

1.3.1 Logging standards

This section will state the complete scenarios for logging in all public methods in the service.

- Method entrance and exit will be logged with DEBUG level.
    - Entrance format: [Entering method {*className.methodName}*]
    - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions. Also note the time of the method call duration.
- Method request and response parameters will be logged with DEBUG level
    - Format for request parameters: [Input parameters[{request_parameter_name_1}:{ request_parameter_value_1}, {request_parameter_name_2}:{ request_parameter_value_2}, etc.)]]
    - Format for the response: [Output parameter {response_value}] . Only do this if there are no exceptions and the return value is not void.
- All exceptions will be logged at ERROR level. This should automatically log inner exceptions as well.
    - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry

3. Log method input parameters
4. If error occurs, log it
5. Log method exit
6. If not void, log method output value
7. Method exit

1.3.2 Error handling

When illegal arguments are encountered, these too will be wrapped in the fault exception. Specifically, any null argument will generate an ArgumentNullException, and any other illegal argument will generate an ArgumentException.

We will take these, and any other exceptions thrown, and convert them into a FaultException, which is done in the following manner:

```
    // Throw converted exception
    throw new FaultException<TCFaultException>
(TCFaultException.CreateFromException(exception));
```

1.3.3 Working with the XSD

The XSD reflects a DataSet, which is why this component uses a DataSet. The general approach is to fill the DataSet with the XML then use it to create client class instances, and also the reverse when returning data. Overall, this is a mapping issue.

1.3.3.1 Mapping Devices

A Device maps to the sensorinfo DataTable in the DataSet.

The mapping for adding/updating is as follows:

- Get site: site:Site = GetSite(username,siteID)
- Get site primary groups: primaryGroups:DeviceGroup[] = deviceGroupRepository.GetForSite(site, DeviceGroup.PRIMARY)
- Get site secondary groups: secondaryGroups:DeviceGroup[] = deviceGroupRepository.GetForSite(site, DeviceGroup.PRIMARY)
- Get primary group: Select from primaryGroups a group whose Name = sensorinfo.HoleNumber
- Get secondary group: Select from secondaryGroups a group whose Name = sensorinfo.LocationType
- Create new DeviceInstall instance
    - Latitude = sensorinfo.X
    - Longitude = sensorinfo.Y
    - InstallationDate = sensorinfo.InService
    - PrimaryGroup = primaryGroup
    - DeviceInstall.SecondaryGroup = secondaryGroup
- Create new Device instance
    - NodeId = sensorinfo.NodeID

- Site = site
- SerialNumber = sensorinfo.SerialNumber
- AddDeviceInstall(deviceInstall)

The mapping for reading into sensorinfo DataTable is as follows. For each Device:

- Get installs for device: installs:DeviceInstall[] = deviceInstallRepository.GetAllForDevice(device)
    - Select the install that is current (i.e. RemovalDate is null)
- HoleNumber = install.PrimaryGroup.Name
- LocationType = install.SecondaryGroup.Name
- X = install.Latitude
- Y = install.Longitude
- InService = install.InstallationDate
- NodeID = device.NodeId
- SerialNumber = device.SerialNumber

1.3.3.2 Mapping DeviceGroups

A device group is transported in the LocationNames DataTable. When adding or updating a DeviceGroup, this table will have 1 entry, and when retrieving, it will have many:

The mapping for adding is as follows:

- Get site: site:Site = GetSite(username,siteID)
- Create new DeviceGroup instance
    - DeviceGroup.Name = LocationNames.Name
    - DeviceGroup.Site = site
    - DeviceGroup.DeviceGroupType = DeviceGroup.PRIMARY

The mapping for update is as follows:

- Get site: site:Site = GetSite(username,siteID)
- Get site device groups: groups:DeviceGroup[] = deviceGroupRepository.GetForSite(site, DeviceGroup.PRIMARY)
- Select group whose name is LocationNames.OldName (Note the developers, this column in this table will be added)
- Update fields in the group
    - DeviceGroup.Name = LocationNames.Name

The mapping for retrieval is as follows:

- Get site: site:Site = GetSite(username,siteID)
- Get site devices: groups:DeviceGroup[] = deviceGroupRepository.GetForSite(site, DeviceGroup.PRIMARY)
- Map each device into the LocationNames row
    - LocationNames.Name = group.Name
    - LocationNames.ID = DeviceGroup.ID

1.3.3.3 Mapping DeviceReadings

The scale parameter determines what type of XSD result data table is used for the result:

- If it is “RealTime", then use the "MinutelyReadings" XSD table.
- If it is “Hourly", then use the "HourlyReadings" XSD table.
- If it is “Daily", then use the "MultiHourlyReadings" XSD table.

Each has the same columns, so the following algorithm applies to either of the tables.

The retrieval of the readings is done for 3 individual reading types. The IReadingRepository.GetReadings(query) method returns a DataTable which has 2 columns: reading_dt (DateTime) and reading_val (double).

As such, we end up with 3 sets of readings, one for each type, and the task is to take the reading of each type at a given time, and create a single row in the XSD result table. No special assumptions can be made about the sorting of the data from the reading repository.

For each row in tempReadings:DataTable

- Get row in percentageReadings:DataTable and saltReadings:DataTable that have the same reading_dt value as the row in tempReadings:DataTable.
- Map to new row in XSD result table
    - Time = reading_dt from the row in tempReadings
    - Percentage = reading_val from the row in percentageReadings
    - Temp = reading_val from the row in tempReadings
    - Salt = reading_val from the row in saltReadings

1.4 Component Class Overview

IDeviceService

This is the interface of the WCF Service for managing devices, device groups, and getting device readings.

DeviceService

This class is the realization of the IDeviceService that implements all methods. It mostly defers to existing client classes to perform the work.

It uses a WebApplicationConfigurationPersistenceProvider to get to its custom configuration in the IIS web.config file. If there are errors during any operation, they are logged using the Logging Wrapper component..

1.5 Component Exception Definitions

DeviceServiceException

This exception extends BaseException. It is used by the IDeviceService to signal a general error. All other exceptions extend this exception.

AuthenticationException
This exception extends DeviceServiceException. It is used by the IUDeviceService to signal authentication failure.

InvalidDataInputException
This exception extends DeviceServiceException. It is used by the IUDeviceService to signal that the data input is not valid.

DeviceServiceConfigurationException
This exception extends DeviceServiceException. It is thrown by the service constructor if anything goes wrong during construction. Such errors would be missing configuration, missing required parameters, etc.

1.6 Thread Safety

The component is effectively thread-safe. This is true in spite of the request entities being mutable, as the service will only work on de-serialized entities, and as such, only one thread will operate on them at a time. Otherwise, the service holds no state, so that all access by multiple threads will not violate thread-safety. Since each call to the client classes is done per client call, thread-safety of those services is not an issue.

The repositories are expected to be thread-safe.

2. Environment Requirements

2.1 Environment

- Development language: C# 3.5
- Compile target: C# 3.5

2.2 TopCoder Software Components

- **Application-Based Configuration 1.1**
    - Provides access to custom configuration sections in the web.config file.
- **Configuration API 1.0**
    - Provides configuration data.
- **Object Factory 1.2**
    - Provides object factory.
- **Object Factory Configuration API Plugin 1.1**
    - Provides object creation facility with Configuration API.
- **Logging Wrapper 3.0.1**
    - Provides the logger used to log activity and errors.

- Use of common logging framework is approved in the forums.

- **WCF Base 2.2**
  - Provides the TCFaultException that is used for exception transport purposes.
- **Exception Manager 2.0**
  - Provides the BaseException.

2.3 Third Party Components

2.3.1 Client schema and source

This component uses the client schema and source code.

3. Installation and Configuration

3.1 Package Name

Toro.TurfGuard.WebService
Toro.TurfGuard.WebService.Impl

3.2 Configuration Parameters

3.2.1 DeviceService

The configuration for this component will be found under one required child object, under the web.config's defaultConfig element, called "device_service_configuration". See demo for an example.

children (required)

Child name	Purpose
object_factory_configuration	To be plugged into the ConfigurationAPIObjectFactory
logging_wrapper_configuration	To be plugged into the LogManager

attributes

property name	value	Required
user_repository_key	Key for the IUserRepository class to pass to ObjectFactory.	Yes
authentication_service_key	Key for the IAuthenticationService class to pass to ObjectFactory.	Yes
device_repository_key	Key for the IDeviceRepository class to pass to ObjectFactory.	Yes
device_group_repository_key	Key for the IDeviceGroupRepository class to pass to ObjectFactory.	Yes
device_install_repository_key	Key for the IDeviceInstallRepository class to pass to ObjectFactory.	Yes
reading_repository_key	Key for the IReadingRepository class to pass to ObjectFactory.	Yes

3.3 Dependencies Configuration

The reader should look at the documentation for each of the component specified in section 2.2 to see how they can be configured.

The logging wrapper is to be configured to use Log4Net

4. Usage Notes

4.1 Required steps to test the component

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

4.2 Required steps to use the component

None.

4.3 Demo

4.3.1 Setup

The demo will operate as if it was performing requests on the WCF services via a client. Since we would use IIS to host, the following configurations would be used:

Create Service.svc file and add a single line in it like this to expose the service to IIS:

```
<%@ ServiceHost="" Language="C#" Debug="true"
Service="Toro.TurfGuard.WebService.Impl.DeviceService" %>
```

The web.config file would contain the following section for this class, using WS biding:

```
  <system.serviceModel>
    <services>
      <service name="Toro.TurfGuard.WebService.Impl.DeviceService">
        <endpoint address=""
contract="Toro.TurfGuard.WebService.Impl.DeviceService"
binding="wsHttpBinding"/>
        <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
      </service>
</services>
    </system.serviceModel>
```

This will assign a default address to the service.

The configuration persistence would be provided with the following child configuration for the service (as per section 3.2.1):

```
<defaultConfiguration>
  <children>
```

```
    <child name="device_service_configuration">
      <properties>
        <property name="user_repository_key">
          <value>user_repository_key</value>
        </property>
        <property name="authentication_service_key">
          <value>authentication_service_key</value>
        </property>
        <property name="device_repository_key">
          <value>device_repository_key</value>
        </property>
        <property name="device_group_repository_key">
          <value>device_group_repository_key</value>
        </property>
        <property name="device_install_repository_key">
          <value>device_install_repository_key</value>
        </property>
        <property name="reading_repository_key">
          <value>reading_repository_key</value>
        </property>

        <!-- The configuration for object factory and logging wrapper.
Not shown for brevity -->
        <child name="object_factory_configuration">
          :
        </child>
        <child name="logging_wrapper_configuration">
          :
        </child>

      </properties>
    </child>

  </children>
</defaultConfiguration>
```

The client would be generated using the *svcutil* tool. The tool would be called in the command prompt as:

```
c:\Svcutil http://localhost:8080/DeviceService.svc?wsdl
/out:DeviceService.cs
```

This would generate a client in the DeviceService.cs file and a corresponding output.xml configuration file. The client can then be instantiated:

```
DeviceServiceClient client = new DeviceServiceClient();
```

All calls will be done in the subsequent section using this client.

4.3.2 *Using the service*

```
// Add device
string deviceXml = // XML with device info
string username = "ivern";
string password = "jimmy";
int siteID = 34;
client.AddDevice(username,password,siteID,deviceXml);

// Update device
```

```
string updatedDeviceXml = // XML with updated device info
client.UpdateDevice(username,password,siteID,updatedDeviceXml);

// Get site devices
string allDevices = client.GetAllDevices(username,password,siteID);

// Add device group
string deviceGroupXml = // XML with device group info
client.AddDeviceGroup(username,password,siteID,deviceGroupXml);

// Update device group
string updatedDeviceGroupXml = // XML with updated device group info
client.AddDeviceGroup(username,password,siteID,updatedDeviceGroupXml);

// Get site device groups
string allDeviceGroups =
client.GetAllDeviceGroups(username,password,siteID);

// Get real-time readings on a device
int readingDepth = 1;
string inputXml = // XML with device info
DateTime startDate = // July 24, 2009
DateTime endDate = // July 29, 2009
string scale = "RealTime";

string reading = client.GetDeviceReading(username, password, siteID,
inputXml, readingDepth, startDate, endDate, scale);
```

5. Future Enhancements

Expansions of the Services to cover a broader range of core entities. Additional performance in the retrieval and transmission of data.