CTC Web Services Generator

Version 1.0.0

Client Tools Consultancy

Table of Contents

1 Introduction

1.1 What is CTC Web Services Generator?

The CTC Web Services Generator is a tool for creating a RESTful Web Services interface for AB Suite systems based on Microsoft Windows Communication Foundation (WCF).

The Generator is an add-in to the Unisys Component Enabler Generate Environment. The generated user interface application utilises the Unisys Component Enabler interface to communicate with AB Suite host systems.

This document should be read in conjunction with the **CTC Web Services Configurator** document and the **CTC Configurator Framework** document.

1.2 The Concept

Typically, a generator creates a fixed, predetermined solution, where users have limited or no influence on what is generated, and customizations must be applied by modifying the generated solution or by writing a custom generator.

The concept of CTC Generators is to include as many requirements as possible in the generate stage rather than applying modifications to the solution after it has been generated.

This requires the generator to be very flexible, and to provide the means for customizing the result of the generator prior to entering the generate phase.

CTC Generators provide the ability to influence what is generated by configuring features, setting options, customizing Standard Data Type Controls, adding Custom Data Type Controls and substituting controls. The generated solution is still based on the forms, fields and controls being defined in the AB Suite development environment, however, the CTC Web Services Configurator allows the developers to specify how each field is to be generated at the application, bundle, language, ispec and field level.

Being able to configure and specify the required customization before the generate phase provides a repeatable and automated solution that in most cases will not need further modifications to the generated source.

In order to provide the necessary flexibility, the Generator includes a library of Standard Data Type Controls, one control for each data type that can be defined in the AB Suite development environment. A Standard Data Type Control is a self-contained type that understands exactly how to interpret the information from the data items and how to create itself as the equivalent Web Service property. This allows the generation of each data item to be easily configured and, when necessary, Custom Data Type Controls can be created by extending the Standard Data Type Controls through inheritance. Custom Data Type Controls can be created to implement custom behaviour.

When the solution is being generated, the generator receives the information from the AB Suite development environment. For each ispec, it creates a Service Model, adding each of the data items to the collection of data items on the Service Model. The Service Model and the data items in the collection generate themselves as the corresponding Web Service properties. During the process, configuration information such as specific data item properties or the replacement of Standard Data Type Controls with Custom Data Type Controls is applied to the data items.

1.3 Standard Data Type Controls

Standard Data Type Controls provide the default behaviour of the data items as they are specified in the AB Suite Development environments. Standard Data Type Controls are built into the Web Services Generator. They can be used as they are without further customization. However, they can also form the basis for customization.

The following is the list of Standard Data Type Controls available with the Web Services Generator:

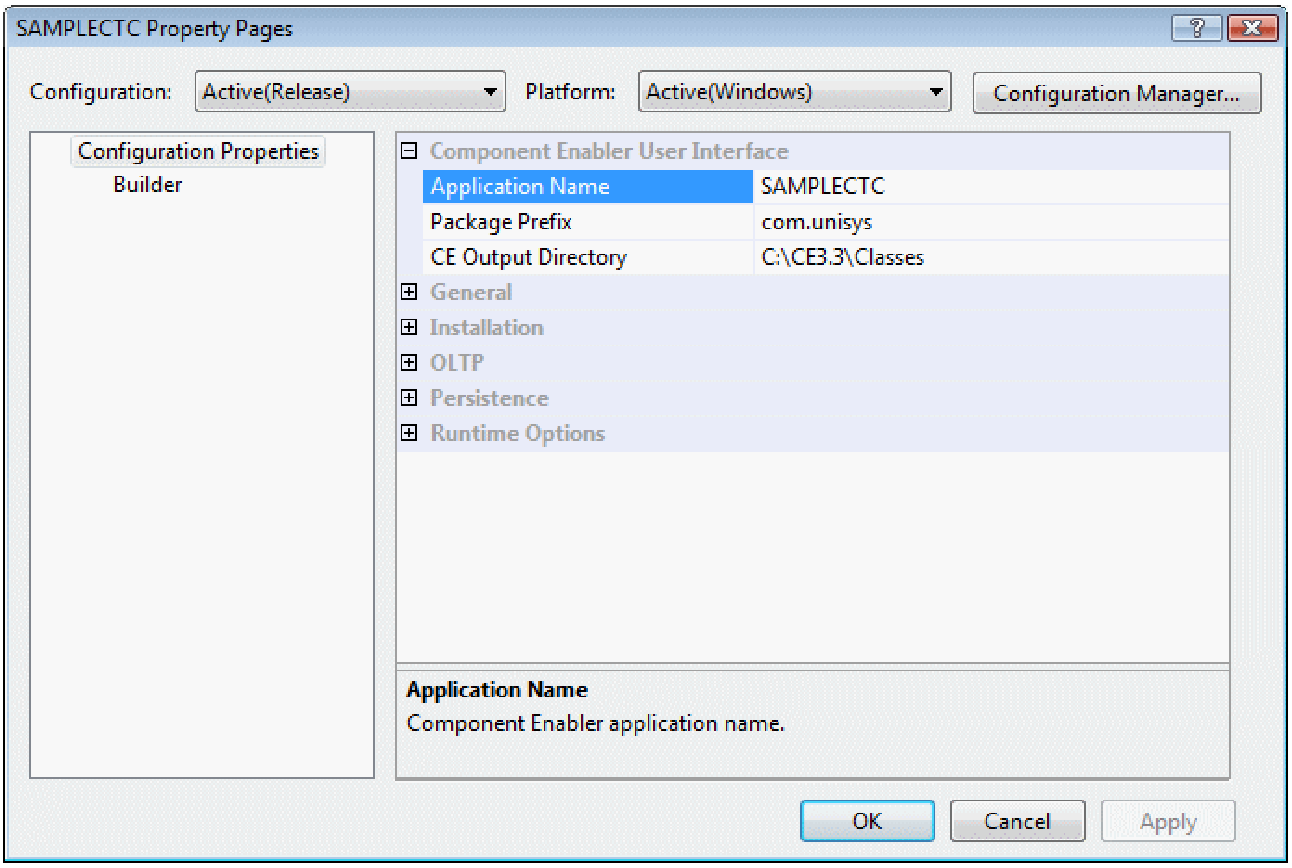
Control	Description
AlphaType	Controls how fields defined as alpha-numeric are generated. This includes Alpha and String types.
BooleanType	Controls how fields painted as single Push Button or single CheckBox that contain true/false values defined at design time are generated.
CopyFromType	Controls how CopyFrom data items are generated.
DateType	Controls how fields defined as Date are generated.
DecimalType	Controls how fields defined as numeric with decimals are generated. This includes Number, Signed, Credit, + and - types.
EnumType	Controls how fields painted as a control that contains a list of values defined at design time are generated. This includes push button, radio button, list box and combo box.
ListType	Controls how fields painted as a list box or combo box creating dynamic lists at runtime are generated.
MaintType	Controls how the Maint field of standard component ispecs is generated.
NumericType	Controls how fields defined as numeric without decimals are generated. This includes Number, Signed, Credit, + and - types.

2 Generator Initial Setup

The CTC Web Services Generator is an add-in generator to the Component Enabler Generate Environment and as such, the setting up of the generator follows the standard instructions for any CE compliant generator.

2.1 AB Suite Setup

Within AB Suite Developer, Application Details must be specified for Component Enabler in the Property Page dialog of the Business Segment Class as shown in the following dialog.



Configuration: Active(Release)
Platform: Active(Windows)

Component Enabler User Interface

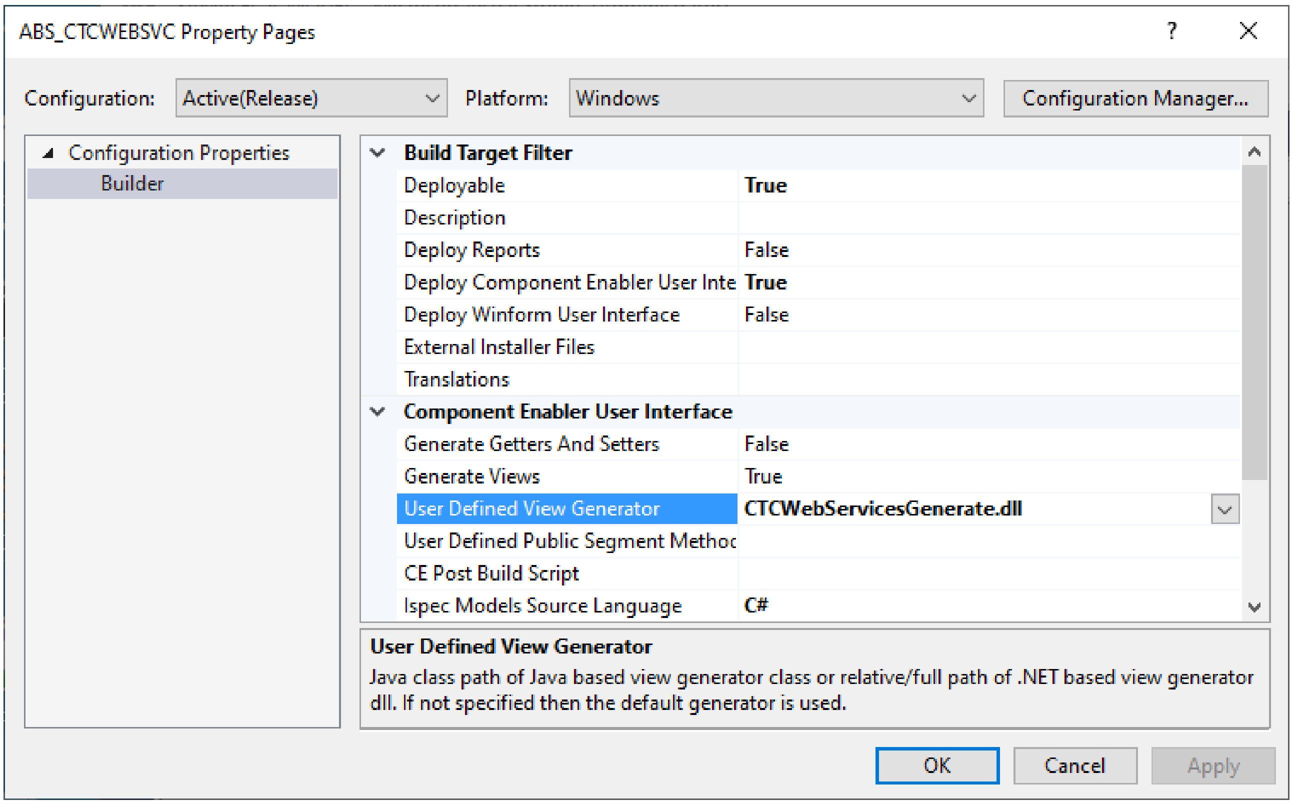
Application Name	SAMPLECTC
Package Prefix	com.unisys
CE Output Directory	C:\CE3.3\Classes

General
Installation
OLTP
Persistence
Runtime Options

Application Name
Component Enabler application name.

Together with the name of the folder defining the bundle as shown below, Application Name, Package Prefix and Output Directory define the path to output location of the generated user interface application. The path is [OutputDirectory]\[PackagePrefix]\[ApplicationName]\[BundleName], i.e. the output location for this example would be C:\CE3.3\Classes\com\unisys\samplectc\inquiry.

A folder defining the ispec classes to include in a bundle is added to the business segment. In the Property Page dialog for the folder, details for the bundle are specified as shown in the following dialog.



Build Target Filter	
Deployable	**True**
Description	
Deploy Reports	False
Deploy Component Enabler User Inte	**True**
Deploy Winform User Interface	False
External Installer Files	
Translations	
Component Enabler User Interface	
Generate Getters And Setters	False
Generate Views	True
User Defined View Generator	**CTCWebServicesGenerate.dll**
User Defined Public Segment Methoc	
CE Post Build Script	
Ispec Models Source Language	**C#**

User Defined View Generator
Java class path of Java based view generator class or relative/full path of .NET based view generator dll. If not specified then the default generator is used.

Deploy Component Enabler User Interface must be set to True.

The name of the CTC Web Services Generator must be specified exactly as shown in User Defined View generator. Generators from CTC are implemented using .NET and the C# language, hence the .dll extension on the name. The reference as specified above is a relative reference to the [ceroot]\bin directory, which is where the generator is located when installed.

Ispec Model Source Language must be set to C#.

For a full description of all fields and how to set up and add ispecs to a folder/bundle, refer to the Unisys AB Suite User Guide.

When building the folder/bundle from AB Suite Developer it is recommended to always choose the 'Rebuild' option in order to ensure the configuration setting of the CTC Generator takes effect on all ispecs in the folder/bundle.

2.2 Installing Bundle Infrastructure Files

To keep the amount of code to be generated to a minimum and to provide a high level of flexibility for customization, the generated solution requires a number of fixed non-generated files to be present in the output directory for compiling and running the application. These files are collectively referred to as Infrastructure Files.

No manual steps are required for copying the infrastructure files to the generate output directory. When running the generator for the first time on a bundle, the infrastructure files are automatically copied into the output directory of the bundle. The files to be copied are controlled by the 'CTCWebServicesInfrastructureFiles.xml' file located in the [ceroot]\bin directory.

When any of the infrastructure files have been updated or new files have been added, the files are re-installed to the bundle by setting the Generator option ReInstallBundle equal True.

When generating a bundle for the first time after upgrading to a new version of the generator, infrastructure files that have changed will automatically be re-installed to the bundle.

3 Configuring the Generator

The generator is configured using the CTC Configurator. The CTC Configurator provides a flexible way of configuring the generator and the controls at any level in a hierarchy consisting of application, bundle, language, ispec and field. The higher in the hierarchy an option or control is configured, the more general it is specified. All lower levels in the hierarchy inherit the specification from the levels above. The lower in the hierarchy an option or control is configured, the more specific it is.

Because the generator may update the configuration file, it is recommended to close the CTC Configurator while running the generator.

For further details on how to use the CTC Configurator, refer to the **CTC Configurator Framework** and **CTC Web Services Configurator** documentation.

3.1 Generator Options

Allow Maint Operations

Allow Maint ADD

This option specifies whether to allow the Add (ADD) function of a Standard Component to be generated as an operation on the service.

Allow Maint BAC

This option specifies whether to allow the Back (BAC) function of a Standard Component to be generated as an operation on the service.

Allow Maint CHG

This option specifies whether to allow the Change (CHG) function of a Standard Component to be generated as an operation on the service.

Allow Maint DEL

This option specifies whether to allow the Delete (DEL) function of a Standard Component to be generated as an operation on the service.

Allow Maint FIR

This option specifies whether to allow the First (FIR) function of a Standard Component to be generated as an operation on the service.

Allow Maint INQ

This option specifies whether to allow the Inquiry (INQ) function of a Standard Component to be generated as an operation on the service.

Allow Maint LAS

This option specifies whether to allow the Last (LAS) function of a Standard Component to be generated as an operation on the service.

Allow Maint NEX

This option specifies whether to allow the Next (NEX) function of a Standard Component to be generated as an operation on the service.

Allow Maint PUR

This option specifies whether to allow the Purge (PUR) function of a Standard Component to be generated as an operation on the service.

Allow Maint REC

This option specifies whether to allow the Recall (REC) function of a Standard Component to be generated as an operation on the service.

Misc Options

Alternate Service Create

This option allows the creation of an alternate service for all or selected ispecs. When set, a copy of the current version of the generated Ispec Service Model is created and added to the project.

Creating an alternate service using the service model of the current generated ispec provides an easy starting point for designing custom services.

Once added, this alternate service is not affected in any way by the generate unless removed using the AlternateServiceRemove option.

Alternate Service Original Remove

When set and an Alternate Service exist for an ispec, the original Ispec ServiceModel will be removed from the generated project. Removing the original Ispec ServiceModel from the project keeps the size of the Adapter dll to a minimum.

Alternate Service Remove

This removes the reference to the alternate service from the generated project. However, in order to preserve the alternate service, the file containing alternate service is not deleted. When later creating an alternate service again for the same ispec, the same file will be added to the generated project.

Build Generated Solution

As part of the generate process, the generator can automatically compile the generated application and build the necessary dll's. This is achieved using MSBuild, which is the build platform from Microsoft used by Visual Studio. Details of the build are written to the generate log file.

Custom Code Module Create

This option allows the creation of a module for all or selected ispecs in which specific custom code can be added. When set for an ispec, the generator adds a code-behind module to the generated Service Model for the ispec and registers this with the generated project. Once created, the module is not affected in any way by the generator unless specifically removed using the 'Custom Code Module Remove' option.

The custom code module allows users to add code to handle specific requirements during the processing of the service. In the custom code module, users can add additional properties to be part of the service message interface or add code for validating input from the external client application before the data is sent to the host system.

Within the Custom Code Module, users decide the implementation of the following two methods which are invoked during the processing of a service operation.

- BeforeTransaction (): This method is invoked before the ispec model is sent to the host system for processing. This occurs after the input request message has been processed and the values have been set on the ispec model. The purpose of this method allows custom code to access fields and properties on the ispec model before it is sent to the host system.
- AfterTransaction (): This method is invoked after the host system has processed the ispec model. This occurs before creating the response message with values from the ispec model. The purpose of this method is to allow custom code to access fields and properties on the ispec model after it is processed by the host system and before a response message is sent to the client application.

Custom Code Module Remove

This removes the reference to the code-behind custom module from the generated project. However, in order to preserve the custom code, the file containing custom code is not deleted. When later creating a custom code module again for the same ispec, the same file will be added to the generated project.

Data Type Decimal

This specifies the client-side type to represent fields defined as numeric with decimals.

This applies to data items defined as Number, Signed, Credit, + and -.

The possible values are: Decimal or Double.

When this is not specified, Decimal is the default.

Data Type Number

This specifies the client-side type to represent fields defined as numeric without decimals.

This applies to data items defined as Number, Signed, Credit, + and -.

The possible values are: Integer or Long.

When this is not specified, Integer is the default.

Default Date Format

This specifies the default date format of date fields used when converting dates between the ispec format and the standard client side DateTime format.

Possible formats are:

- UK (ddMMyy, ddMMyyyy)
- US (MMddyy, MMddyyyy)
- International (yyMMdd, yyyyMMdd)

When this is not specified, UK is the default.

Display Errors, Display Warnings

The user can be alerted to errors and warnings that occur during the generate process. By default, errors and warnings are displayed in a message box waiting for confirmation. Regardless of the setting of these options, errors and warnings are always written to the generate log file.

Generate System Info

This specifies for the generator to write information about ispecs and fields contained in the bundle being generated to the CTCSystemInfo.xml file located in the [ceroot]/bin directory. This information is used by the CTC Configurator to populate the dropdowns on fields, ispecs, bundles and applications with valid selections making it easy to configure these items.

IIS Reset

When set, the generator resets (Stop and Start) IIS before generating Ispec Model files. This avoids errors occurring during the generation of the Ispec Model files caused by IIS locking Ispec Model files that have previously been accessed.

Include List Data in Response

This option specifies whether to include the content of lists in the response message for fields represented as ListBox or ComboBox that create a list of data at runtime. When not specified, the default is ‘False’.

Include Usage Input in Response

This specifies whether to include fields defined as Usage Input in the response message. When not specified, the default is ‘False’.

Include Usage Inquiry in Response

This specifies whether to include fields defined as Usage Inquiry in the response message. When not specified, the default is ‘True’.

Include Usage IO in Response

This specifies whether to include fields defined as Usage IO in the response message. When not specified, the default is ‘True’.

Infrastructure Files Version Check

When re-initializing a bundle, this option determines whether to perform version check when copying infrastructure files to the bundle output directory. When set, only new and updated files are copied.

List of Values As Enum

This option specifies whether to create an Enum type for fields represented by controls such as PushButton, RadioButton, ListBox or ComboBox for which a list of values are defined.

When enabled, the list of values specified at design time for these controls will be generated as an Enum type allowing the client application to only select from the Enum list.

Log Level

This defines the level of detail written to the generate log file. The log information is written to C:\Temp\Generate.log.

Re-Install Bundle

When set this option causes the generator to re-install the infrastructure files to the bundle output directory. This is required when new files have been added, when existing files have been updated or to repair damaged files. The re-install is performed when next starting the generate process of the bundle. When done, this option is automatically reset by the generator.

Single Button as Boolean

This option specifies whether to create a Boolean type for fields represented by a single control such as PushButton and CheckBox that include true/false values.

For PushButtons, the action value represents the true value. The false value for a PushButton is always blank.

For CheckBoxes, the checked value represents the true value and the unchecked value represents the false value.

Two Digit Year Cutoff

This option specifies an integer from 1 to 9999 that represents the cut off year for interpreting two-digit years as four-digit years. This option is used when converting a 6-digit date to a language neutral date in the format yyyymmdd.

A two-digit year that is less than or equal to the last two digits of the cut off year is in the same century as the cut off year. A two-digit year that is greater than the last two digits of the cut off year is in the century that precedes the cut off year. For example, if two-digit year cut off is 2056 (the default), the two-digit year 56 is interpreted as 2056 and the two-digit year 57 is interpreted as 1957. In other words, a two-digit year cut off of 2056 specifies dates in the 100 years range between 1957 and 2056. This is the equivalent of the Base Year specified on the Business Segment of EAE and AB Suite, which has a default value of 1957.

Use Data Display as Name

This specifies whether to use the data display text of fields for creating meaningful property names when creating the Service Model.

Duplicated values will have a number appended in order to create unique names.

Use Ispec Description as Name

This specifies whether to use the description of ispecs for creating meaningful service names.

Duplicated values will have a number appended in order to create unique names.

Note: When using the CustomCodeModuleCreate option and enabling/disabling the UseIspecDescriptionAsName option, the service name of the custom code module will not be changed. That is because the generator does not modify the custom code module after it has been created the first time.

Virtual Directory Auto Create

This option configures the generator to automatically create a virtual directory for the generated Web Services solution. On machines without IIS, this option should set to false.

Virtual Directory Name New

When initializing a new bundle, the generator automatically creates a virtual directory for the generated web application on the local host. By default, the name of the virtual directory is [ApplicationName]_[BundleName]. This parameter allows assignment of a new name. The name is changed when next generating the bundle.

Visual Studio Version

This option specifies the Visual Studio version. This is currently a fixed value 'VS2008' which cannot be changed.

The generator will automatically copy infrastructure files appropriate to the chosen value to the bundle views directory. This includes files such as project files for the generated solution.

Naming Operations

Name Maint ADD

This specifies the name of an Add (ADD) operation. By default, the name of the ADD operation is 'Add'.

Name Maint BAC

This specifies the name of a Back (BAC) operation. By default, the name of the BAC operation is 'Prior'.

Name Maint CHG

This specifies the name of a Change (CHG) operation. By default, the name of the CHG operation is ‘Change’.

Name Maint DEL

This specifies the name of a Delete (DEL) operation. By default, the name of the DEL operation is ‘Delete’.

Name Maint FIR

This specifies the name of a First (FIR) operation. By default, the name of the FIR operation is ‘First’.

Name Maint INQ

This specifies the name of an Inquiry (INQ) operation. By default, the name of the INQ operation is ‘Inquiry’.

Name Maint LAS

This specifies the name of a Last (LAS) operation. By default, the name of the LAS operation is ‘Last’.

Name Maint NEX

This specifies the name of a Next (NEX) operation. By default, the name of the NEX operation is ‘Next’.

Name Maint PUR

This specifies the name of a Purge (PUR) operation. By default, the name of the PUR operation is ‘Purge’.

Name Maint REC

This specifies the name of a Recall (REC) operation. By default, the name of the REC operation is ‘Recall’.

Name Memo Operation

This specifies the name of a Memo operation. By default, the name of the memo operation is ‘Transaction’.

Naming Types

Name Alternate Type

This specifies the suffix name of an Alternate Service Model type to be appended to the ispec name in order to create a unique type name.

By default, the Alternate type name is ‘Alt’.

Name CopyFrom Type

This specifies the suffix name of a CopyFrom type that is appended to the ispec name in order to create a unique type name.

By default, the CopyFrom type name is 'RepeatingGroup'.

Name Input Type

This specifies the suffix name of an Input type that is appended to the ispec name in order to create a unique type name.

An Input type is created for operations such as Add, Change and Memo when the Operations Input Type is Object.

By default, the Input type name is 'Input'.

Name Key Type

This specifies the suffix name of a Key type that is appended to the ispec name in order to create a unique type name.

A Key type is created for Maint operations such as Back, Delete, Inquiry, Next, Purge and Recall that take keys as input when the Operations Input Type is Object.

By default, the Key type name is 'Key'.

Name List Type

This specifies the suffix name of a List type that is appended to the field name in order to create a unique type name.

By default, the List type name is 'List'.

Name Output Type

This the suffix name of an Output type that is appended to the ispec name in order to create a unique type name.

An Output type is created for operations such as Back, First, Inquiry, Last, Next, Recall and Memo.

By default, the Output type name is 'Output'.

Name Status Type

This specifies the suffix name of a Status type that is appended to the ispec name in order to create a unique type name.

A Status type is created for Maint operations such as Add, Change, Delete and Purge that only returns status information from the transaction.

By default, the Status type name is 'Status'.

3.2 Data Type Control Specifications

For each field type available in AB Suite, the generator creates a property on the generated Service Model. Each of the field types can be configured using the CTC Configurator to customize the default Data Type control template.

The table below shows the default Data Type Control template for each of the data types.

Parameters in square brackets (e.g.: [PROPERTYNAME]) represent values that will be replaced at generated time with values specified in the AB Suite Development environment.

Alpha Type Control

Controls how fields defined as alpha-numeric are generated. This includes Alpha and String types.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="[DATATYPE]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type. Default type is String.

Boolean Type Control

Controls how fields painted as single Push Button or single CheckBox that contain true/false values defined at design time are generated.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="ClientDataTypes.BooleanType" TrueValue="[TRUEVALUE]"
FalseValue="[FALSEVALUE]" />
<!--
Specify TrueValue and FalseValue attributes.
TrueValue and FalseValue are used for converting ispec field value
to/from Boolean.

When ispec field value matches the TrueValue, true is sent to the
client.
When ispec field value matches the FalseValue, false is sent to
the client.
When ispec field value does not match either TrueValue or
FalseValue, false is sent to the client.

When true is received from the client, the ispec field is set to
the TrueValue.
When false is received from the client, the ispec field is set to
the FalseValue.
-->
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type.

TrueValue: The value of the data item that represents 'true'.

FalseValue: The value of the data item that represents 'false'.

CopyFrom Type Control

Controls how CopyFrom data items are generated.

Control Specifications:

```
<CopyFromItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
CopyFromType="[COPYFROMTYPE]" Comparison="[COMPARISON]"
ComparisonField="[COMPARISONFIELD]"
ComparisonValue="[COMPARISONVALUE]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

CopyFromType: The type name of the CopyFrom type.

Comparison: Specifies a comparison test to perform at runtime to determine whether a copyfrom row is to be included in the response message. This allows eliminating blank rows or other specific copyfrom rows. The test compares the ComparisonValue against the value of the ComparisonField. The copyfrom row is included in the response message when the comparison is true.

Possible values are: Equal, NotEqual, GreaterThan, LessThan and None. None means no comparison is performed and all rows are included in the response message.

ComparisonField: The field to compare against.

ComparisonValue: The value to compare against the ComparisonField.

By default, no comparison is performed and all CopyFrom rows are included in the client response message.

By default, the ComparisonField will be set to the first field in the CopyFrom row.

Date Type Control

Controls how fields defined as Date are generated.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="[DATATYPE]" DateFormat="[DATEFORMAT]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type. Default type is DateTime.

DateFormat: Possible values are UK, US and International.

Dates are converted to a language neutral format yyyymmdd when copied to the web services response object.

Decimal Type Control

Controls how fields defined as numeric with decimals are generated. This includes Number, Signed, Credit, + and - types.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="[DATATYPE]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type. Default type is Double or Decimal as defined by the DataTypeDecimal option.

Enum Type Control

Controls how fields painted as a control that contains a list of values defined at design time are generated. This includes push button, radio button, list box and combo box.

Control Specifications:

```
<EnumItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
EnumType="[ENUMTYPE]">
    [ENUMS]
<!--
To customize the enumerations, replace the [ENUMS] parameter
with one or more Enum elements formatted as below
<Enum Key="xxxx" Value="yyyy" Default="true" />
 - Key is the host system value.
 - Value is the value of the enum.
 - Default identifies the enum that is to be the default.
-->
</EnumItem>
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

EnumType: The type name of the Enum type.

By default, the list of label/value pairs specified at design time on PushButton, RadioButton, ListBox and ComboBox will be generated as Enum elements to the [ENUMS] parameter.

However, enumeration keys and values can be customized by replacing the [ENUMS] parameter with one or more Enum elements.

List Type Control

Controls how fields painted as a list box or combo box creating dynamic lists at runtime are generated.

Control Specifications:

```
<ListItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
ReturnEmptyList=="[RETURNEMPTYLIST]" ListType="[LISTTYPE]"
XmlFilePath="" XmlElementPath="">
    [LISTCOLUMNS]
<!--
To customize the list columns, replace the [LISTCOLUMNS] parameter
with a Key element and one or more Column elements formatted as
below:
<Key Order="" PropertyName="PropertyName of Key"
DataType="ClientDataTypes.StringType" Column="1" />
<Column Order="" PropertyName="PropertyName of columnX"
DataType="ClientDataTypes.StringType" Column="X" />
-->
</ListItem>
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

ReturnEmptyList: When true, an empty list is returned when the list does not exist. When false, an error is returned when the list does not exist. Default value is false.

ListType: The type name of the List type.

XmlFilePath: Specifies an external XML formatted file to send to the client as a list. When specified, the file will automatically be loaded and included in the response to the client. The path to the file is relative to the 'views' directory of the generated bundle. For example: XmlFilePath="ExternalLists/Countries.xml" points to views\ExternalLists\Countries.xml

XmlElementPath: This is an XPath expression specifying the path to the xml element within the file holding the data to include in the list. The path must be specified relative to the root element. For example: XmlElementPath="//country" specifies the country element within the root.

By default, Key and Column elements identifying the columns will be generated with default values to the [LISTCOLUMNS] parameter. The key column is named 'Key' and the columns are named 'Column1' - 'ColumnN'. The data type of each column is String.

The Column property allows specifying substring. I.e. Column="2(0-10)" specifies to take 10 characters from column 2 of the list from the host starting at index 0.

However, the column specifications can be customized by replacing the [LISTCOLUMNS] parameter with individual Key and Column elements.

Maint Type Control

Controls how the Maint field of standard component ispecs is generated.

Control Specifications:

```
<IspecMaintField PropertyName="[PROPERTYNAME]"
IspecFieldName="[ISPECFIELDNAME]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

IspecFieldName: The property name of the Maint field.

Note: The Maint field is never part of the request/response message. The Maint field is set indirectly by invoking an operation.

Numeric Type Control

Controls how fields defined as numeric without decimals are generated. This includes Number, Signed, Credit, + and - types.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="[DATATYPE]" />
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type. Default type is Integer or Long as defined by the DataTypeNumeric option.

See the **CTC Web Services Configurator** documentation for further details.

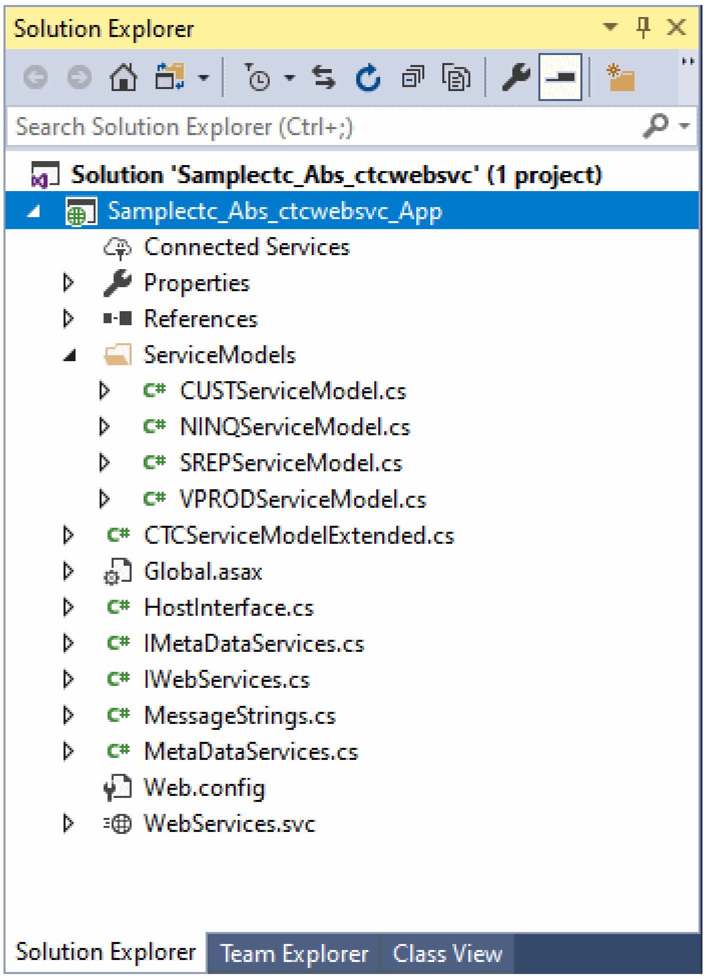
See also **Section 9, Custom Data Type Controls** for information about additional Custom Data Type controls.

4 The Generated Web Services Solution

The generated solution is based on the Microsoft Windows Communication Foundation (WCF), which provides a very flexible solution for exposing ispecs as RESTful Web Services.

4.1 The Visual Studio Solution

The generated solution is created for Visual Studio 2013, 2015, 2017 or 2019. The solution can be opened and inspected using Visual Studio. Below is an example of a generated solution with four ispecs - CUST, NINQ, SREP and VPROD – from the Sample system in the bundle.



```
Solution 'Samplectc_Abs_ctcwebsvc' (1 project)
  Samplectc_Abs_ctcwebsvc_App
    Connected Services
    Properties
    References
    ServiceModels
      CUSTServiceModel.cs
      NINQServiceModel.cs
      SREPServiceModel.cs
      VPRODServiceModel.cs
    CTCServiceModelExtended.cs
    Global.asax
    HostInterface.cs
    IMetaDataServices.cs
    IWebServices.cs
    MessageStrings.cs
    MetaDataServices.cs
    Web.config
    WebServices.svc
```

The folder named **ServiceModels** contains a generated file for each ispec included in the bundle. A Service Model contains the Meta data information required by the CTC Web Service Controller to expose an ispec as a Web service. For further information about the Service Models, see section below.

The file WebServices.svc provides the interface between the client application calling a web services and the CTC Web Service Controller module and the AB Suite backend host system. WebServices.svc is invoked when the client application the makes a call to a web service and the call is then directed to the appropriate ispec on the AB Suite system.

The CTC Web Service Controller manages the execution of the operations of the ispec Service Models and manages all communication with the AB Suite host system. For further information about the CTC Web Service Controller, see section below.

HostInterface file allows customizing the execution of the web services such as PreTransaction, PostTransaction, StatusLine and CustomConnect. For further information see the CTC Web Services Controller section below.

MetaDataServices file is used by the CTC Web Services Test Client for getting the necessary meta data information about the web services and their operations when creating the user interface for the Testing the Web Services.

4.2 The Generated Service Models

A Service Model class is generated for each ispec. A Service Model contains the Meta data information required by the CTC Web Service Controller to expose an ispec as a Web Service. The Meta data includes information such as:

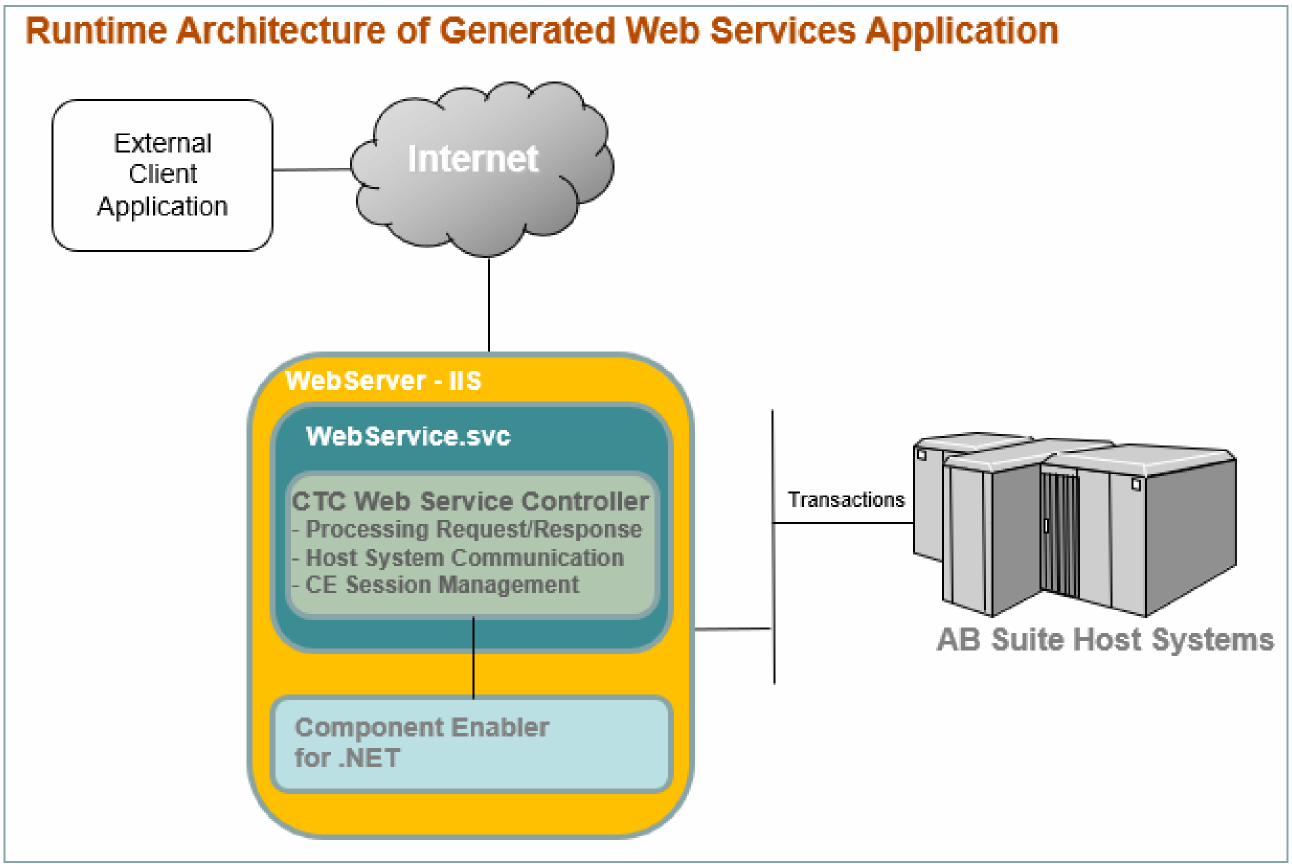
- Ispec name and type
- Operations available on the ispec
- Data items on the ispec and their attributes such as type, read only and write only
- CopyFrom type describing the data items and their attributes in a CopyFrom row
- List types describing the columns and their attributes in a list
- Enum types describing the enumerations and their attributes

Service Models and their Meta data are used:

- When running the CTC Web Services Test Client for creating the service operation signatures (method input/output) and properties depending on the service being tested.
- At runtime by the CTC Web Service Controller when processing a request message and creating a response message.

5 CTC Web Service Controller

The CTC Web Service Controller is a generic control that manages all communication to the AB Suite backend host systems. It is part of the Adapter Connection class from which access to an EAE/AB Suite host system is required. The default HostInterface.cs file delivered with the installation of the generator, provides an example of how to use the Service Controller and provides a starting point for customizing the Service Controller.


Runtime Architecture of Generated Web Services Application
External Client Application
Internet
WebServer - IIS
WebService.svc
CTC Web Service Controller
- Processing Request/Response
- Host System Communication
- CE Session Management
Component Enabler for .NET
Transactions
AB Suite Host Systems

The diagram above provides an overview of the runtime architecture of a Web Services generated by the CTC Web Services Generator. It illustrates how the CTC Web Service Controller fits into the architecture as an essential component and shows the major functions it performs.

The CTC Web Service Controller processes the incoming request message from the external client application, creates a session with the host system, sends the transaction to the host system and creates the response message to be returned to the client application.

The Web Service Controller uses the standard Component Enabler from Unisys to communicate with the AB Suite host system.

The CTC Web Service Controller can be configured for using connection pooling providing an effective connection and session management to the host system. See Web.config file for further information.

The HostInterface.cs file hosting the CTC Web Service Controller can be customized to control the behaviour of the Service Controller by using the programmatic interface. The HostInterface can get control at key events during the execution of service operations and the communication with the AB Suite host system. The Service Controller raises the following events:

- PreTransaction – Occurs after processing the request message from the Client application and before the transaction is sent to the host system. This provides the opportunity for the adapter to check the data before it is sent to the host system and to bypass the transaction or to cancel the transaction. Setting the BypassTransaction property on the event arguments causes the transaction to be ignored by the Service Controller and not to be sent to the host system.

- PostTransaction – Occurs after the host system has responded to the transaction and before creating the response message. This provides the opportunity for the adapter to check the data before it is returned to the Client application.
- StatusLine – Occurs just before creating and returning the response message to the Client application. This provides the opportunity for the adapter to check the status of the transaction, to check the status line information returned by the host system and to customize the error handling.
- CustomConnect – Occurs when the session with the host system needs to be connected. This provides the opportunity for the adapter to customize the connection to the host system.

For examples of how to customize these event handlers, see the HostInterface.cs file in the generated solution.

6 Deployment

6.1 Deploy Generated Solution

The generated solution includes a DeploySolution.bat file located in the Views folder. This bat file provides details of the files required to be copied to the deployment environment.

6.2 Setting Configuration Parameters

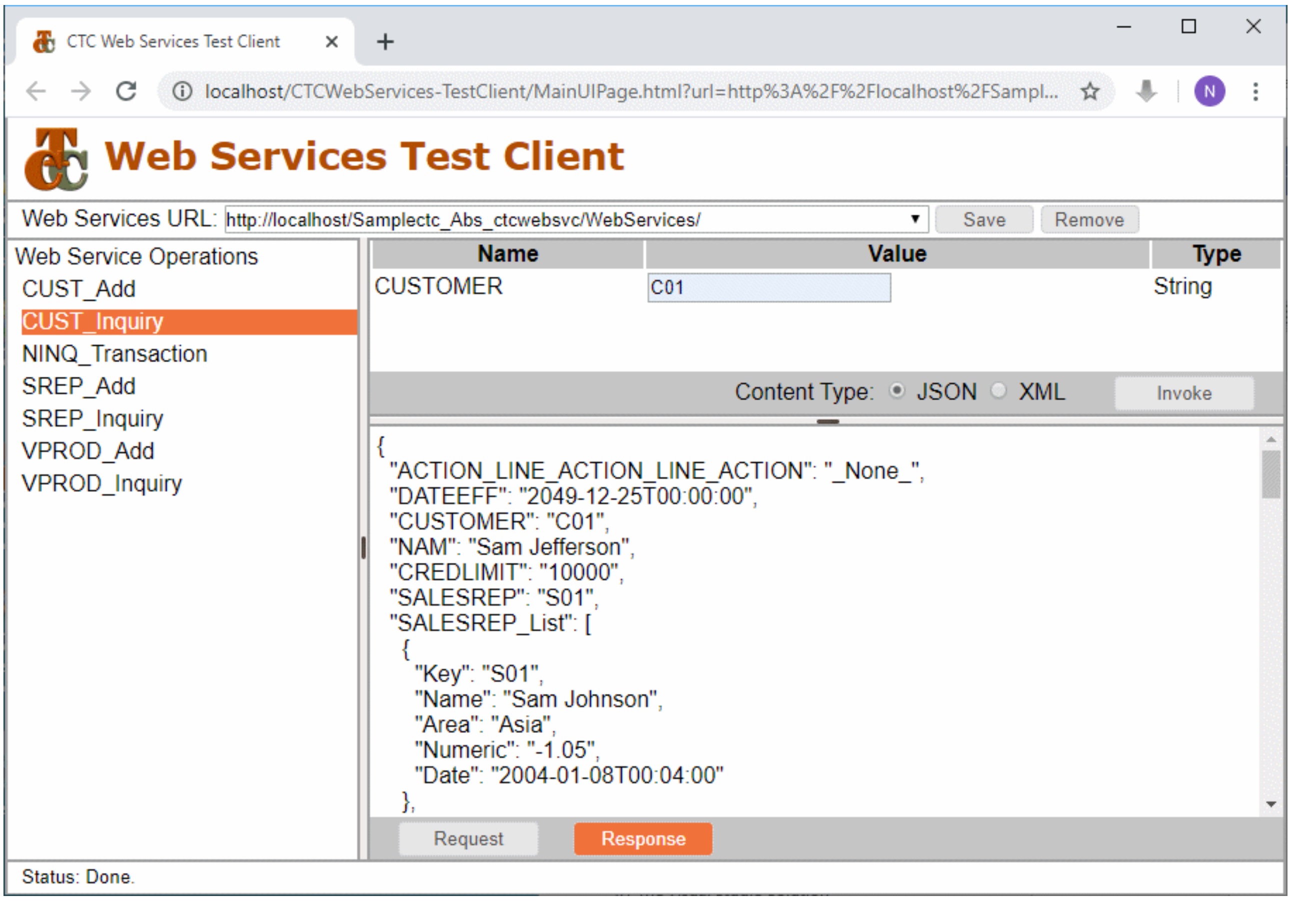
Runtime configuration parameters are specified in the standard Web.config file located in the generated solution folder. Open the Web.config file in a text editor and edit the parameters as appropriate for your environment. The Web.config parameters are the standard Component Enabler parameters.

The System.ServiceModel section is required for the operation of the generated service models. Parameters in this section has been pre-set and should not be changed.

7 Test Client

Installed with the generator is a Test Client, which is a tool for testing the generated web services without having to create and develop a Client Application.

When generating a bundle/folder for the first time, a short cut for starting the Test Client with either IE or Chrome is added to the views folder. To run the Test Client, double click on the short cut for your preferred browser.



Name	Value	Type
CUSTOMER	C01	String

```
{
 "ACTION_LINE_ACTION_LINE_ACTION": "_None_",
 "DATEEFF": "2049-12-25T00:00:00",
 "CUSTOMER": "C01",
 "NAM": "Sam Jefferson",
 "CREDLIMIT": "10000",
 "SALESREP": "S01",
 "SALESREP_List": [
  {
   "Key": "S01",
   "Name": "Sam Johnson",
   "Area": "Asia",
   "Numeric": "-1.05",
   "Date": "2004-01-08T00:04:00"
  },
```

As seen in the above example, the Test Client shows a list of operations available on the generated Web Services. Selecting an operation, input fields required for the operation are created for each field on the ispec model. When entering data in the fields and submitting the request by clicking the Invoke button, the response from the web services is shown. Clicking the Request button shows the URI, HTTP Request method, Content-Type and Data format of the request for the web service operation.

8 Consuming the Generated Web Services

Having generated the Web Services, they are then ready to be used from any environment capable of sending and receiving messages across the internet. This include mobile devices, desktop computers, Java environments, Microsoft.NET environments and browser environments with JavaScript.

In the following, the steps for using the Web Services from a .NET application and a browser JavaScript application are outlined.

8.1 Web Service Operations

The interface to the generated web services include the following generic operations:

- Inquiry
  - This operation is used for inquiring on a standard ispec component
  - Uri Template format: /Inquire/{ispecname}?key1=value1&…&keyN=valueN
  - Http method: GET

   - Key fields to identify record to inquire on are passed as http query parameters
   - Sample URI for invoking the operation:
     http://machinename/[VirtualDirName]/WebServices/Inquire/CUST?Customer=C01
- Add
   - This operation is used for adding a record to a standard ispec component
   - Uri Template format: /Add/{ispecname}
   - Http method: POST
   - Input fields are passed as http POST parameters
   - Sample URI for invoking the operation:
     http://machinename/[VirtualDirName]/WebServices/Add/CUST
- Update
   - This operation is used for updating a record on a standard ispec component
   - Uri Template format: /Update/{ispecname}
   - Http method: POST
   - Input fields are passed as http POST parameters
   - Sample URI for invoking the operation:
     http://machinename/[VirtualDirName]/WebServices/Update/CUST
- Delete
   - This operation is used for deleting a record on a standard ispec component
   - Uri Template format: /Delete/{ispecname}?key1=value1&…&keyN=valueN
   - Http method: DELETE
   - Key fields to identify record to delete are passed as http query parameters
   - Sample URI for invoking the operation:
     http://machinename/[VirtualDirName]/WebServices/Delete/CUST?Customer=C01
- Purge
   - This operation is used for purging a record on a standard ispec component
   - Uri Template format: /Purge/{ispecname}?key1=value1&…&keyN=valueN
   - Http method: DELETE
   - Key fields to identify record to purge are passed as http query parameters
   - Sample URI for invoking the operation:
     http://machinename/[VirtualDirName]/WebServices/Purge/CUST?Customer=C01
- Navigate
   - This operation is used for navigating (FIR, BAC, NEX or LAS) on a standard ispec component
   - Uri Template format: /Navigate/{ispecname} /{function}?key1=value1&…&keyN=valueN
   - Http method: GET
   - Key fields to identify record to navigate on are passed as http query parameters

  - Sample URI for invoking the operation:
    http://machinename/[VirtualDirName]/WebServices/Navigate/NEX/CUST?Customer=C01
- Transaction
  - This operation is used for transacting with a memo ispec component
  - Uri Template format: /Transaction/{ispecname}
  - Http method: POST
  - Input fields are passed as http POST parameters
  - Sample URI for invoking the operation:
    http://machinename/[VirtualDirName]/WebServices/Transaction/NINQ

These operations are defined in the IWebServices.cs and WebServices.svc.cs files. If required, these files can be customized to modify the generic operations or include additional services.

8.2 Invoking Web Services from a .NET Console Application

The following shows a simple example of calling a Web Service from a .NET console application built using Visual Studio. The same steps apply to any .NET application such as Console, Windows and WPF from which access to web services is required.

The examples are using C# and the .NET WebClient class for invoking web services operations. For information on .NET WebClient see https://docs.microsoft.com/en-us/dotnet/api/system.net.webclient?view=netcore-3.1

The example below invokes the Inquiry operation on the CUST ispec retrieving customer C01.

```
using System;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Net;

namespace SampleExternalApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Invoke Customer Inquiry Web Service
            string webServiceURI =
"http://localhost/Samplectc_Abs_ctcwebsvc/WebServices/Inquire/CUST?customer=C01";

            // Call network methods in a try/catch block to handle exceptions.
            try
            {
                WebClient client = new WebClient();

                // Type of receiving data. Default is json.
                //client.Headers[HttpRequestHeader.Accept] = "application/json";
                //client.Headers[HttpRequestHeader.Accept] = "application/xml";

                // Invoke the web service.
                Stream data = client.OpenRead(webServiceURI);
                // Read the response.
                StreamReader reader = new StreamReader(data);
                string response = reader.ReadToEnd();

                data.Close();
                reader.Close();

                // Process the response here.
                // Check errorcode field in response for transaction error.
                Console.WriteLine(response);

                // Keep the console window open.
                Console.WriteLine("Press any key to exit.");
                Console.ReadKey();
            }
            catch (Exception e)
            {
                Console.WriteLine("\nException Caught!");
                Console.WriteLine("Message :{0} ", e.Message);
                // Keep the console window open.
                Console.WriteLine("Press any key to exit.");
                Console.ReadKey();
            }
        }
    }
}
```

```
Result (JSON format):
{
  "ACTION_LINE_ACTION_LINE_ACTION": "",
  "DATEEFF": "2049-12-25T00:00:00",
  "CUSTOMER": "C01",
  "NAM": "Sam Jefferson",
  "CREDLIMIT": "10000",
  "SALESREP": "S01",
  "CUST_TYPE": "A",
  "POSTADD1": "PO Box 1234",
  "POSTADD2": "North Adelaide",
  "POSTADD3": "South Australia 5006",
  "DELADD1": "29 PARK TERRACE",
  "DELADD2": "Clearview",
  "DELADD3": "South Australia 5085",
  "S_NAME": "Sam Johnson",
  "TransactionResponseStatus": {
    "Messages": [
      "11:46:43:68 INQUIRY REQUEST"
    ]
  },
  "errorcode": 0
}

Sample error result (JSON format):
{
  "errorcode": 911,
  "fault": {
    "faultcode": "Sender",
    "faultstring": "The transaction failed. Check faultdetails for error message(s).
(CE response code: 911)",
    "faultdetails": "Customer Number ** ITEM NOT FOUND **"
  }
}

Result (XML format):
<?xml version="1.0" encoding="utf-8"?>
```

```
<response>
  <ACTION_LINE_ACTION_LINE_ACTION />
  <DATEEFF>2049-12-25T00:00:00</DATEEFF>
  <CUSTOMER>C01</CUSTOMER>
  <NAM>Sam Jefferson</NAM>
  <CREDLIMIT>10000</CREDLIMIT>
  <SALESREP>S01</SALESREP>
  <CUST_TYPE>A</CUST_TYPE>
  <POSTADD1>PO Box 1234</POSTADD1>
  <POSTADD2>North Adelaide</POSTADD2>
  <POSTADD3>South Australia 5006</POSTADD3>
  <DELADD1>29 PARK TERRACE</DELADD1>
  <DELADD2>Clearview</DELADD2>
  <DELADD3>South Australia 5085</DELADD3>
  <S_NAME>Sam Johnson</S_NAME>
  <TransactionResponseStatus>
    <Messages>
      <Value>11:52:58:46 INQUIRY REQUEST</Value>
    </Messages>
  </TransactionResponseStatus>
  <errorcode>0</errorcode>
</response>

Sample error result (XML format):
<response>
  <errorcode>911</errorcode>
  <fault>
    <faultcode>Sender</faultcode>
    <faultstring>The transaction failed. Check faultdetails for error message(s). (CE
response code: 911)</faultstring>
    <faultdetails>Customer Number ** ITEM NOT FOUND **</faultdetails>
  </fault>
</response>
```

The example below invokes the Add operation on the CUST ispec to add a new customer C99.

```
using System;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Net;

namespace SampleExternalApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Invoke Customer Add Web Service
            string webServiceURI =
"http://localhost/Samplectc_Abs_ctcwebsvc/WebServices/Add/CUST";

            // Data to send as a string (JSON type).
            string newCustomer = "{" +
                "\"DATEEFF\": \"2020-12-01\"," +  // Send date in Universal Date Format
                "\"ACTION_LINE_ACTION_LINE_ACTION\": \"\"," +
                "\"CUSTOMER\": \"C99\"," +
                "\"NAM\": \"New Name\"," +
                "\"CREDLIMIT\": \"9000\"," +
                "\"SALESREP\": \"S01\"," +
                "\"CUST_TYPE\": \"A\"," +
                "\"POSTADD1\": \"12 Postal St\"," +
                "\"POSTADD2\": \"New Postal\"," +
                "\"POSTADD3\": \"New State 1234\"," +
                "\"DELADD1\": \"12 Delivery St\"," +
                "\"DELADD2\": \"New Delivery\"," +
                "\"DELADD3\": \"New State 1234\"" +
            "}";
            /*
            // Data to send as a string (XML type).
            string newCustomer = "<request>" +
                "<DATEEFF>2020-12-01</DATEEFF>" +  // Send date in Universal Date
Format
                "<ACTION_LINE_ACTION_LINE_ACTION></ACTION_LINE_ACTION_LINE_ACTION>" +
                "<CUSTOMER>C99</CUSTOMER>" +
                "<NAM>New Name</NAM>" +
                "<CREDLIMIT>9000</CREDLIMIT>" +
                "<SALESREP>S01</SALESREP>" +
                "<CUST_TYPE>A</CUST_TYPE>" +
                "<POSTADD1>12 Postal St</POSTADD1>" +
                "<POSTADD2>New Postal</POSTADD2>" +
                "<POSTADD3>New State 1234</POSTADD3>" +
                "<DELADD1>12 Delivery St</DELADD1>" +
                "<DELADD2>New Delivery</DELADD2>" +
                "<DELADD3>New State 1234</DELADD3>" +
            "</request>";
            */

            // Call synchronous network methods in a try/catch block to handle
exceptions.
            try
            {
                WebClient client = new WebClient();

                // Type (JSON or XML) of sending data. Default is json.
                //client.Headers[HttpRequestHeader.ContentType] = "application/json";
                //client.Headers[HttpRequestHeader.ContentType] = "application/xml";

                // Type of receiving data. Default is json.
                //client.Headers[HttpRequestHeader.Accept] = "application/json";
                //client.Headers[HttpRequestHeader.Accept] = "application/xml";

                // Invoke the web service.
                string response = client.UploadString(webServiceURI, "POST",
newCustomer);

                // Process the response here.
                // Check errorcode field in response for transaction error.
                Console.WriteLine(response);

                // Keep the console window open.
                Console.WriteLine("Press any key to exit.");
                Console.ReadKey();
            }
            catch (Exception e)
            {
                Console.WriteLine("\nException Caught!");
                Console.WriteLine("Message :{0} ", e.Message);
                // Keep the console window open.
                Console.WriteLine("Press any key to exit.");
                Console.ReadKey();
            }
        }
```

```
    }
}
Result (JSON format):
{
  "TransactionResponseStatus": {
    "Messages": [
      "12:19:31:65 SUCCESSFUL ENTRY      000001"
    ]
  },
  "errorcode": 0
}

Sample error result (JSON format):
{
  "errorcode": 911,
  "fault": {
    "faultcode": "Sender",
    "faultstring": "The transaction failed. Check faultdetails for error message(s).
(CE response code: 911)",
    "faultdetails": "Customer Number ** ITEM ALREADY EXISTS **"
  }
}

Result (XML format):
<?xml version="1.0" encoding="utf-8"?>
<response>
  <TransactionResponseStatus>
    <Messages>
      <Value>12:24:27:77 SUCCESSFUL ENTRY      000001</Value>
    </Messages>
  </TransactionResponseStatus>
  <errorcode>0</errorcode>
</response>

Sample error result (XML format):
<response>
  <errorcode>911</errorcode>
  <fault>
    <faultcode>Sender</faultcode>
    <faultstring>The transaction failed. Check faultdetails for error message(s). (CE
response code: 911)</faultstring>
    <faultdetails>Customer Number ** ITEM ALREADY EXISTS **</faultdetails>
  </fault>
</response>
```

8.3 Invoking Web Services from a browser JavaScript Application

The following shows a simple example of calling a Web Service from a browser JavaScript application using the JQuery AJAX method for invoking web services operations.

The example below invokes the Inquiry operation on the CUST ispec retrieving customer C01.

```
MainUIPage.prototype._getCust = function () {
    var self = this;

    var url =
"http://localhost/Samplectc_Abs_ctcwebsvc/WebServices/Inquire/CUST?Customer=C01";
var dataType = "json";    // Type (JSON or XML) of receiving data
```

```
    $.ajax({
        type: "GET",
        url: url,
        success: function (response) {
            // Check for transaction error.
            if (response.errorcode !== 0) {
                alert(response.fault.faultstring + "\n\n" +
response.fault.faultdetails);
            }
            else {
                // Success. Process the response object.
                // Show response object in pretty format.
                alert("Response object:\n" + JSON.stringify(response, undefined, 4));
            }
        },
        dataType: dataType
    });
};
```

```
Result (JSON format):
Response object:
{
    "ACTION_LINE_ACTION_LINE_ACTION": "",
    "DATEEFF": "2049-12-25T00:00:00",
    "CUSTOMER": "C01",
    "NAM": "Sam Jefferson",
    "CREDLIMIT": "10000",
    "SALESREP": "S01",
    "CUST_TYPE": "A",
    "POSTADD1": "PO Box 1234",
    "POSTADD2": "North Adelaide",
    "POSTADD3": "South Australia 5006",
    "DELADD1": "29 PARK TERRACE",
    "DELADD2": "Clearview",
    "DELADD3": "South Australia 5085",
    "S_NAME": "Sam Johnson",
    "TransactionResponseStatus": {
        "Messages": [
            "12:52:29:65 INQUIRY REQUEST"
        ]
    },
    "errorcode": 0
}

Sample error result (JSON format):
{
    "errorcode": 911,
    "fault": {
        "faultcode": "Sender",
        "faultstring": "The transaction failed. Check faultdetails for error
message(s). (CE response code: 911)",
        "faultdetails": "Customer Number ** ITEM NOT FOUND **"
    }
}
```

The example below invokes the Add operation on the CUST ispec to add a new customer C99.

```
MainUIPage.prototype._addCust = function () {
    var self = this;
    var newCustomer = {
        "DATEEFF": "2020-12-01", // Send date in Universal Date Format
        "ACTION_LINE_ACTION_LINE_ACTION": "",
        "CUSTOMER": "C99",
        "NAM": "New Name",
        "CREDLIMIT": "9000",
        "SALESREP": "S01",
        "CUST_TYPE": "A",
        "POSTADD1": "12 Postal St",
        "POSTADD2": "New Postal",
        "POSTADD3": "New State 1234",
        "DELADD1": "12 Delivery St",
        "DELADD2": "New Delivery",
        "DELADD3": "New State 1234"
    };

    var url = "http://localhost/Samplectc_Abs_ctcwebsvc/WebServices/Add/CUST";
    var contentType = "json"; // Type of sending data
    var dataType = "json";    // Type of receiving data

    $.ajax({
        type: "POST",
        url: url,
        data: JSON.stringify(newCustomer), // Send data as a string
        success: function (response) {
            // Check for transaction error.
            if (response.errorcode !== 0) {
                alert(response.fault.faultstring + "\n\n" +
response.fault.faultdetails);
            }
            else {
                // Success. Process the response object.
                // Show response object in pretty format.
                alert("Response object:\n" + JSON.stringify(response, undefined, 4));
            }
        },
        contentType: contentType,
        dataType: dataType
    });
};
```

```
Result (JSON format):
Response object:
{
    "TransactionResponseStatus": {
        "Messages": [
            "13:00:59:06 SUCCESSFUL ENTRY      000001"
        ]
    },
    "errorcode": 0
}
Sample error result (JSON format):
{
    "errorcode": 911,
    "fault": {
        "faultcode": "Sender",
        "faultstring": "The transaction failed. Check faultdetails for error
message(s). (CE response code: 911)",
        "faultdetails": "Customer Number ** ITEM ALREADY EXISTS **"
    }
}
```

9 Custom Data Type Controls

Custom Data Type Controls are controls that extend a Standard Data Type Control to implement specific requirements.

Custom Data Type Controls are used for substituting Standard Data Type Controls on the generated Service Model when requirements demand functionality that cannot be satisfied by the Standard Data Type Controls. Using the CTC Configurator, Standard Data Type Controls can be substituted with Custom Data Type Controls (see the **CTC Web Services Configurator** documentation for further details).

Included with the CTC Web Services Generator are a number of Custom Data Type Controls. These can be used out of the box or changed to suit local requirement.

9.1 System Provided Custom Data Type Controls

Below is a list of Custom Data Type Controls that are delivered as part of the CTC Web Services Generator.

Boolean Custom Type Control

A Boolean Custom Type Control allows representing a data item to the client application as a boolean and would be used when the value of a data item can only be one of two different values.

Control Specifications:

```
<IspecDataItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"
DataType="ClientDataTypes.BooleanType" TrueValue="A"
FalseValue="B" />
<!--
Specify TrueValue and FalseValue attributes.
TrueValue and FalseValue are used for converting ispec field value
to/from boolean.

When ispec field value matches the TrueValue, true is sent to the
client.
When ispec field value matches the FalseValue, false is sent to
the client.
When ispec field value does not match either TrueValue or
FalseValue, false is sent to the client.

When true is received from the client, the ispec field is set to
the TrueValue.
When false is received from the client, the ispec field is set to
the FalseValue.
-->
```

Order: Specifies the order the data item appears in the request/response message.

PropertyName: The property name of the data item.

DataType: The client-side data type.

TrueValue: The value of the data item that represents ‘true’.

FalseValue: The value of the data item that represents 'false'.

Enum Custom Type Control
 	
An Enum Custom Type Control allows representing a data item to the client application as an enum type and would be used when the value of a data item can only be one of a list of values.
 	
Control Specifications:
`<EnumItem Order="[ORDER]" PropertyName="[PROPERTYNAME]"`
`EnumType="[ENUMTYPE]">`
`<Enum Key="xxxx" Value="yyyy" Default="true" />`
`<!--`
`To customize the enumerations, insert one or more Enum elements.`
` - Key is the host system value.`
` - Value is the value of the enum.`
` - Default identifies the enum that is to be the default.`
`-->`
`</EnumItem>`
Order: Specifies the order the data item appears in the request/response message.
 	
PropertyName: The property name of the data item.
 	
EnumType: The type name of the Enum type.
 	
Insert an Enum element for each possible value of the data item:
 	
Key: The value of the data item on host system data base.
 	
Value: The value on the client side.
 	
Default: When 'true', this identifies the enum item to be the default when no match is found.

9.2 Creating Own Custom Data Type Controls

Custom Data Type Controls can be created to implement specific requirements when none of the Standard Data Type Controls cover the requirements.

Custom Data Type Controls can be implemented easily without the need to customize the whole generator. A Custom Data Type Control is created as a class using Visual Studio and when implemented, it is added to the generator using the CTC Configurator.

Once added to the generator, Custom Data Type Controls can then extend or substitute controls on the generated Service Models. Using the CTC Configurator, data types on the Service Model can be configured to be substituted with Custom Data Type Controls and the condition for when to do the substitution. For further details, see the **CTC Web Services Configurator** documentation.

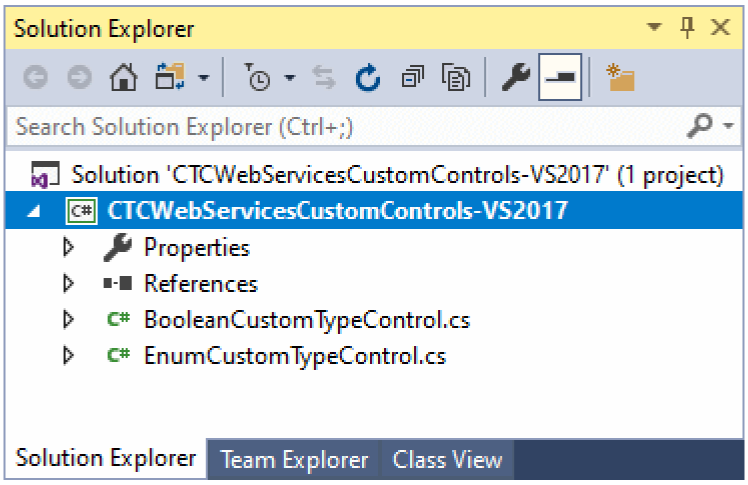
When creating Custom Data Type Controls, there are two distinct areas that need to be considered:

- The Generate Side
- The Runtime Side

9.2.1 Custom Data Type Controls – The Generate Side

Included with the installation of the CTC Web Services Generator is a sample Visual Studio project that provides two examples of how to implement the Generate Side of Custom Data Type Controls.

The project is named **CTCWebServicesCustomControls.csproj** and is installed into the **CustomControls** directory of the **[ceroot]\CTC-Software** folder. The project is shown below.



Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'CTCWebServicesCustomControls-VS2017' (1 project)
CTCWebServicesCustomControls-VS2017
Properties
References
BooleanCustomTypeControl.cs
EnumCustomTypeControl.cs

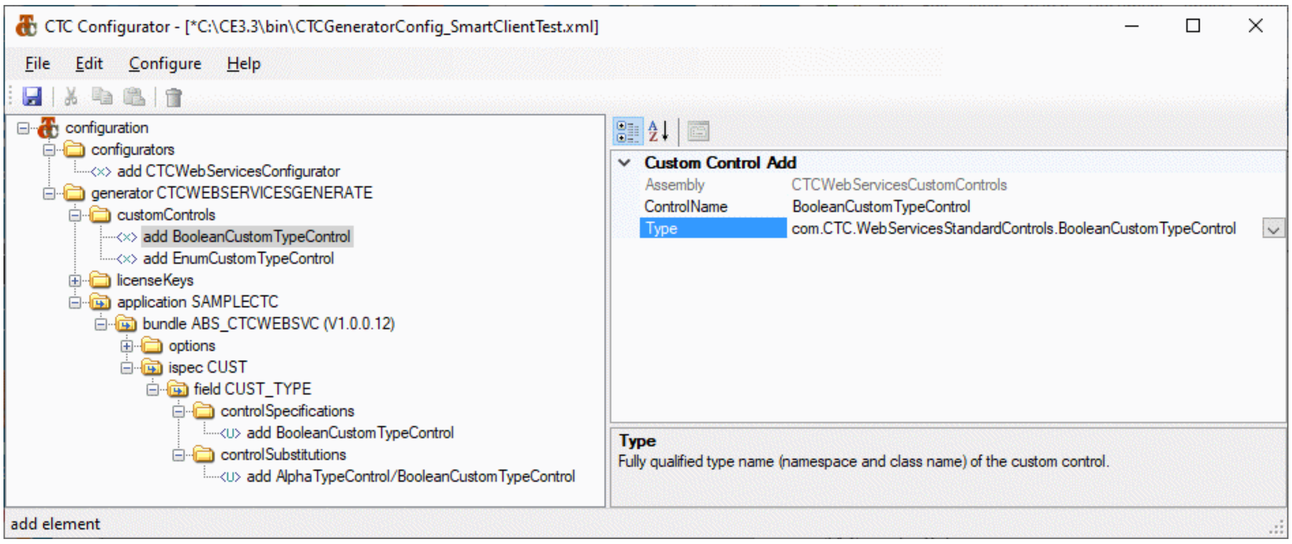
Solution Explorer | Team Explorer | Class View

When building the CTCWebServicesCustomControls.csproj, a CTCWebServicesCustomControls.dll is created and added to the bin directory of the [ceroot] folder. It may be necessary to modify the CTCWebServicesCustomControls.csproj to point to the [ceroot] directory on the local machine.

The BooleanCustomType control is an example of how to implement a Custom Data Type Control. The BooleanCustomType class implements the Generate Side and extends the CTC Standard AlphaType Control to re-use as much of the generation of the AlphaType control as possible. For the generation of a control, one method needs to be implemented:

- RenderControlSpecifications()
  This method outputs the specifications of the control to the ServiceModel.cs file. The specification defines how the property of the data item is generated.

The following is an example of the BooleanCustomType control when added to the generator using the CTC Configurator.



CTC Configurator - [*C:\CE3.3\bin\CTCGeneratorConfig_SmartClientTest.xml]

```
configuration
  configurators
    add CTCWebServicesConfigurator
  generator CTCWEBSERVICESGENERATE
    customControls
      add BooleanCustomTypeControl
      add EnumCustomTypeControl
    licenseKeys
    application SAMPLECTC
      bundle ABS_CTCWEBSVC (V1.0.0.12)
        options
        ispec CUST
          field CUST_TYPE
            controlSpecifications
              add BooleanCustomTypeControl
            controlSubstitutions
              add AlphaTypeControl/BooleanCustomTypeControl
```

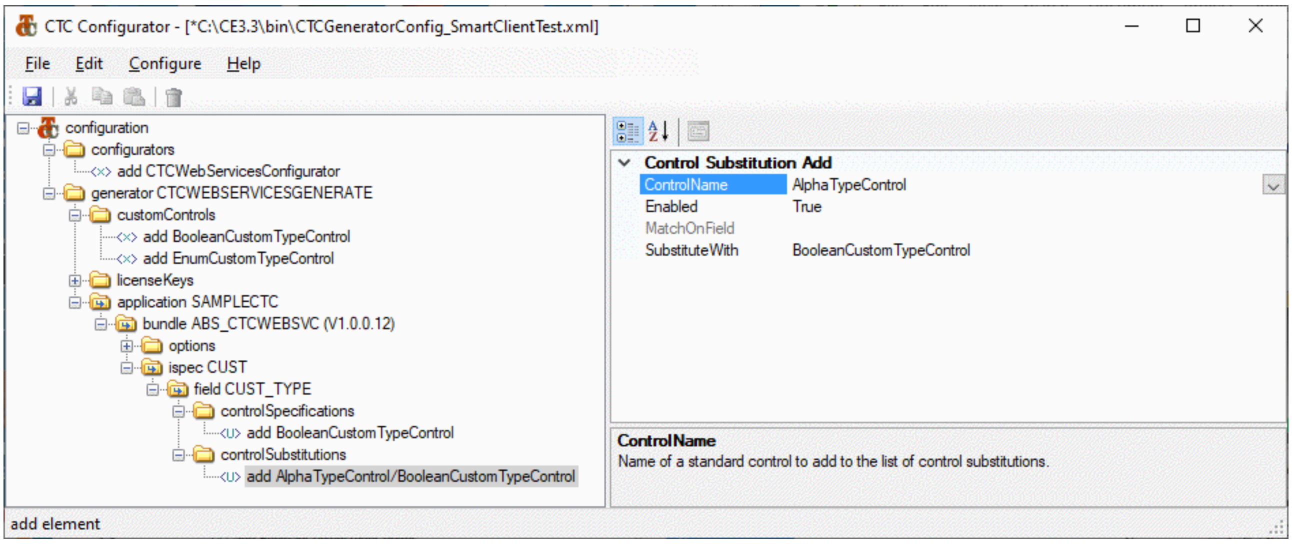
Custom Control Add

Assembly	CTCWebServicesCustomControls
ControlName	BooleanCustomTypeControl
Type	com.CTC.WebServicesStandardControls.BooleanCustomTypeControl

Type
Fully qualified type name (namespace and class name) of the custom control.

add element

Below is an example of the BooleanCustomType when substituting the standard AlphaType control with the BooleanCustomType using the CTC Configurator.



CTC Configurator - [*C:\CE3.3\bin\CTCGeneratorConfig_SmartClientTest.xml]

```
configuration
  configurators
    add CTCWebServicesConfigurator
  generator CTCWEBSERVICESGENERATE
    customControls
      add BooleanCustomTypeControl
      add EnumCustomTypeControl
    licenseKeys
    application SAMPLECTC
      bundle ABS_CTCWEBSVC (V1.0.0.12)
        options
        ispec CUST
          field CUST_TYPE
            controlSpecifications
              add BooleanCustomTypeControl
            controlSubstitutions
              add AlphaTypeControl/BooleanCustomTypeControl
```

Control Substitution Add

ControlName	AlphaTypeControl
Enabled	True
MatchOnField	
SubstituteWith	BooleanCustomTypeControl

ControlName
Name of a standard control to add to the list of control substitutions.

add element

9.2.2 Custom Data Type Controls - The Runtime Side

If specific runtime behaviour of a Custom Data Type Control is required, it can be implemented on the CTCServiceModelExtended class.