DocGeneration Software Development Kit (SDK)

Last updated: 10th July 2015

Table of Contents

Building Web Service Data Sources

The Qorus DocGeneration framework allows developers to construct their own web service data sources which expose a set of data tags and/or data tables and their values when called by Qorus DocGeneration. This is done using a standard set of .NET 3.5 interfaces and utilities. These web services can have optional input parameters, and need to return an array of object arrays which will become the set of data tags and/or data table tags available to the Qorus Generation template.

Web Method Structure

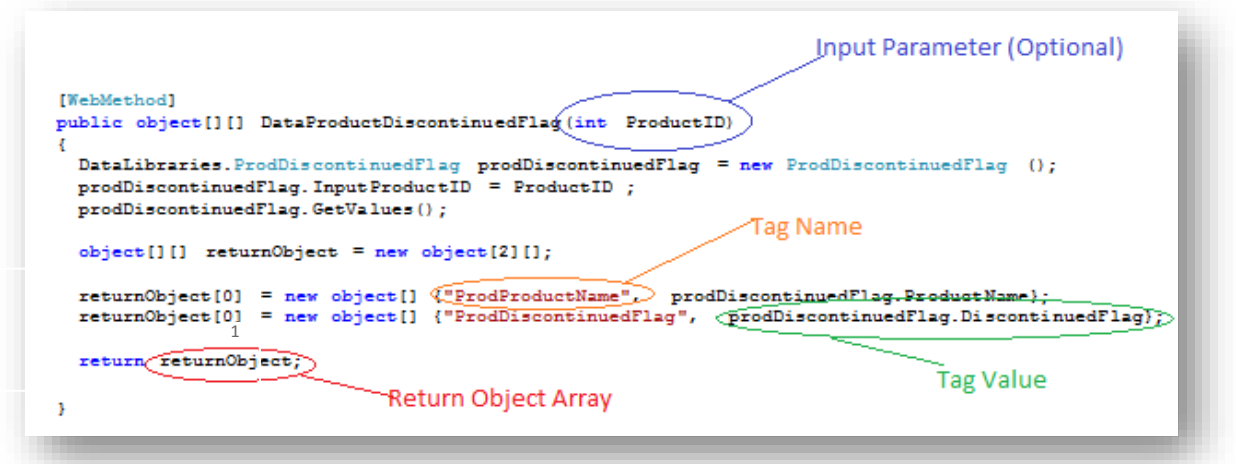


Figure 1: Data Source Web Service

```
CODE SNIPPET

      [WebMethod]
      public object[][] DataProductDiscontinuedFlag(int ProductID)
      {
        DataLibraries.ProdDiscontinuedFlag prodDiscontinuedFlag = new ProdDiscontinuedFlag ();
        prodDiscontinuedFlag.InputProductID = ProductID ;
        prodDiscontinuedFlag.GetValues();

        object[][] returnObject = new object[2][];

        returnObject[0] = new object[] {"ProdProductName", prodDiscontinuedFlag.ProductName};
        returnObject[1] = new object[] {"ProdDiscontinuedFlag",
prodDiscontinuedFlag.DiscontinuedFlag};

        return returnObject;
      }
```

- The web service's web method can have no or multiple input parameters.
- The method needs to return an array of object arrays, made up of Data Tag Name and Data Tag Value elements. The Data Tag Value type is preserved, so Qorus Generation will pick up the right Data Tag type once the Web Service Data Source is created in the template.
- Data tag field names need to be unique within a Qorus Generation Template.
- Images need to be streamed to a byte array in order to be returned as an image data tag
- Tables need to be serialised to a byte array in order to be returned as data table tag.

```
CODE SNIPPET

      [WebMethod]
      public object[][] EmpNameByReportsTo(int managerEmployeeID)
        {
            DataLibraries.EmployeesNameByReportsTo employeesNameByReportsTo = new
EmployeesNameByReportsTo();
            employeesNameByReportsTo.ReportsToID  = managerEmployeeID ;
            employeesNameByReportsTo.GetValues();

            object[][] returnObject = null ;

            if (employeesNameByReportsTo.EmployeeDataTable != null)
            {
                DataTable dTable = employeesNameByReportsTo.EmployeeDataTable;
                string data = SerialiseDataTable(dTable);
                object[][] obj = { new object[] { "EmployeeDataTable", data } };
                returnObject = obj;
            }

            return returnObject;
      }
```

- Here is an example piece of code that accepts a DataTable type, and serialises it to a byte array.

```
private string SerialiseDataTable(DataTable dTable)
{
    MemoryStream str = new MemoryStream();
    dTable.WriteXml(str, XmlWriteMode.WriteSchema);
    byte[] Bytes = str.ToArray();
    return Encoding.UTF8.GetString(Bytes);
}
```

Figure 2: Serialise Data Table

```
CODE SNIPPET

 private string SerialiseDataTable(DataTable dTable)
  {
       MemoryStream str = new MemoryStream();
       dTable.WriteXml(str, XmlWriteMode.WriteSchema);
       byte[] Bytes = str.ToArray();
       return Encoding.UTF8.GetString(Bytes);
  }
```

- In order to create a data source to use in a repeating section, a nested data table structure is required. If you wish to nest a data table into another, you will need to have a column of type string in the top level data table that contains a serialised data table.
- An important requirement for nested table structures is that the tag that you assign to a table has that same name as the table that it returns. Otherwise no data will be returned when the template that uses the nested repeating section with the new data source is merged.

```
CODE SNIPPET

[WebMethod]
public object[][] EmpDetailsByReportsTo(int managerEmployeeID)
{
   DataTable dTable = new DataTable("EmployeesDetailsTable");
   dTable.Columns.Add("EmployeeId", System.Type.GetType("System.Int32"));
   dTable.Columns.Add("EmployeeAge", System.Type.GetType("System.Int32"));
   dTable.Columns.Add("EmployeeSalesTable", System.Type.GetType("System.String"));
   dTable.Columns.Add("EmployeeAgentsTable", System.Type.GetType("System.String"));

   DataLibraries.EmployeesDetailsByReportsTo employeesDetailsByReportsTo = new
EmployeesDetailsByReportsTo();
   employeesDetailsByReportsTo.ReportsToID = managerEmployeeID;
   employeesDetailsByReportsTo.GetValues();

   dTable.Rows.Add(employeesDetailsByReportsTo.EmployeeId, employeesDetailsByReportsTo.EmployeeAge,
                           SerialiseDataTable(employeesDetailsByReportsTo.EmployeeSalesTable),
                           SerialiseDataTable(employeesDetailsByReportsTo.EmployeeAgentsTable));

   object[][] returnObject = null;
    returnObject = new object[dTable.Rows.Count][];
    string data = SerialiseDataTable(dTable);
    object[][] obj = { new object[] { "dataTable", data } };
    returnObject = obj;

    return returnObject;
}
```

Building 'On Completion' Web Services

The Qorus DocGeneration framework allows a developer to build a web service that can be called once a Qorus DocGeneration template has been merged. This is done using a standard set of .NET 3.5 interfaces and utilities. The web method allows the document to be manipulated further depending on user requirements.

Qorus DocGeneration 'On Completion' Namespace URL

On Completion web services are required to use the following namespace URL (case sensitive): *http://qorus.oncompletion.org/*

```
[WebService(Namespace = "http://qorus.oncompletion.org/")]
```

Figure 3: On Completion Namespace URI

Web Method Structure

```
using System.Runtime.Serialization.Formatters.Binary;


namespace OnCompletionWebService
{
    /// <summary>
    /// Summary description for OCService_Dummy
    /// </summary>
    [WebService(Namespace = "http://qorus.oncompletion.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class OCService_Dummy : System.Web.Services.WebService
    {
        [WebMethod]
        public void OnCompletion(byte[] document, byte[] tags)
        {
            string msg = "Byte Array Length: " + document.Length.ToString() + " || Number of input fields: " +
                CommonCode.DeserializeArrayOfStringArray(tags).Length.ToString();
        }
    }
}
```

Required Namespace URI

Required Web Method Format

Input Tags: Serialized Array of String Arrays

Figure 4: On Completion Web Service

```
CODE SNIPPET
……..
using System.Runtime.Serialization.Formatters.Binary;

namespace OCServiceLoggingService
{
    /// <summary>
    /// Summary description for OCServiceLoggingService
    /// </summary>
    [WebService(Namespace = "http://qorus.oncompletion.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]

    public class OCServiceLoggingServiceLogMerges : System.Web.Services.WebService
    {
        [WebMethod]
        public void OnCompletion(byte[] document, byte[] tags)
        {
            string msg = "File Length: " + document.Length.ToString() + " || number of input data
tags: " + CommonCode.DeserialiseArrayOfStringArray(tags).Length.ToString();
            CommonCode.LogToSystem("OCServiceLoggingServicesLogMerges", msg, document, tags);
         }
    }
}
```

- An ‘On Completion’ web method requires the following:
  - No return value
  - To be named “OnCompletion”
  - To have only the parameters *byte[] document* and *byte[] tags*
- The *document* byte array contains a byte stream of the merged document
  - The *tags* byte array contains a serialised array of string arrays, and these string arrays are name/value pairs of all the data tags used to merge the template. The first 6 tags contained by the array are:
  - The URL in which the triggering item resides: "*OriginatingListURL*"
  - The SiteID of the site in which the triggering item resides: "*OriginatingSPSiteID*"
  - The WebID of the site in which the triggering item resides: "*OriginatingSPWebID*"
  - The ListID of the site in which the triggering item resides: "*OriginatingSPListID*"
  - The ItemUniqueID of the site in which the triggering item resides: "*OriginatingSPItemUniqueID*"
  - The merged document name: "*DocumentName*"
- In order to extract the data tags, use the following example code to de-serialise the array.

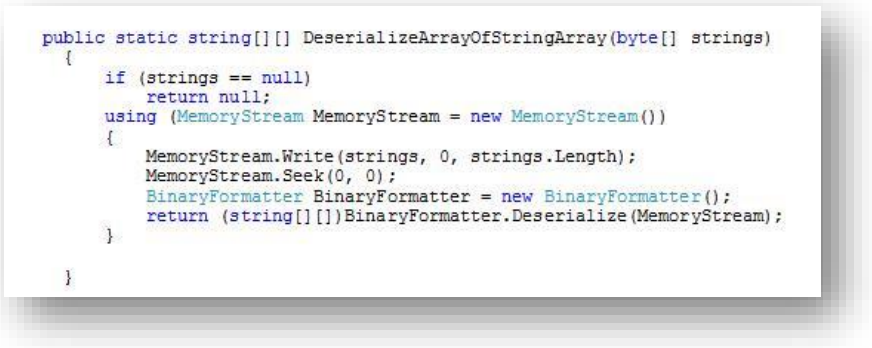


Figure 5: De-Serialise Array of String Arrays

CODE SNIPPET

```
        public static string[][] DeserialiseArrayOfStringArray(byte[] strings)
        {
            if (strings == null)
                return null;
            using (MemoryStream MemoryStream = new MemoryStream())
            {
                MemoryStream.Write(strings, 0, strings.Length);
                MemoryStream.Seek(0, 0);
                BinaryFormatter BinaryFormatter = new BinaryFormatter();
                return (string[][])BinaryFormatter.Deserialise(MemoryStream);
            }
        }
```

Configure 'On-completion' Web Service

When setting up an on-completion web service event for a template, the user has the option to configure the specific web service settings. For example, if your on-completion service is designed to send emails, this function could be used to allow the user to enter a specific 'To' address or subject for the email.

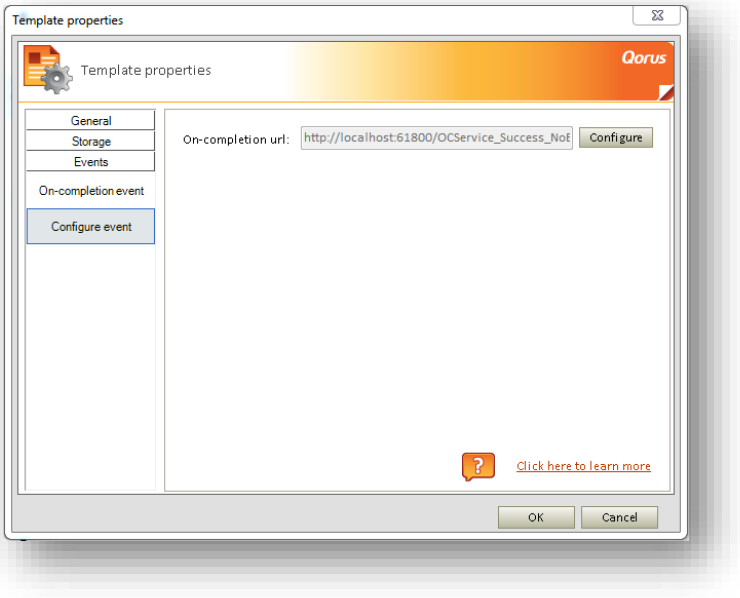


Figure 6: On-completion Event Screen without configuration

In order to allow the user to configure your web service settings, you'll need to create an .ASPX page with the same name as your web service .ASMX. This page will then be loaded into the template properties screen.

i.e. Where your Web Service is named OCServiceLoggingService.asmx your configure page needs to be named; *OCServiceLoggingService.aspx*

The configuration page can save data against the template; this can then be used at a later stage when the on-completion service is executed. To allow for this, your aspx page is required to include a javascript function "OnCompletionConfigXml". This function should return a string, which will represent the configuration data that is saved against your template. The template properties dialog will invoke this method when the configuration page is performs a submit/post action. The configuration returned by the method will then be saved against the template and can be retrieved at a later stage by the API.

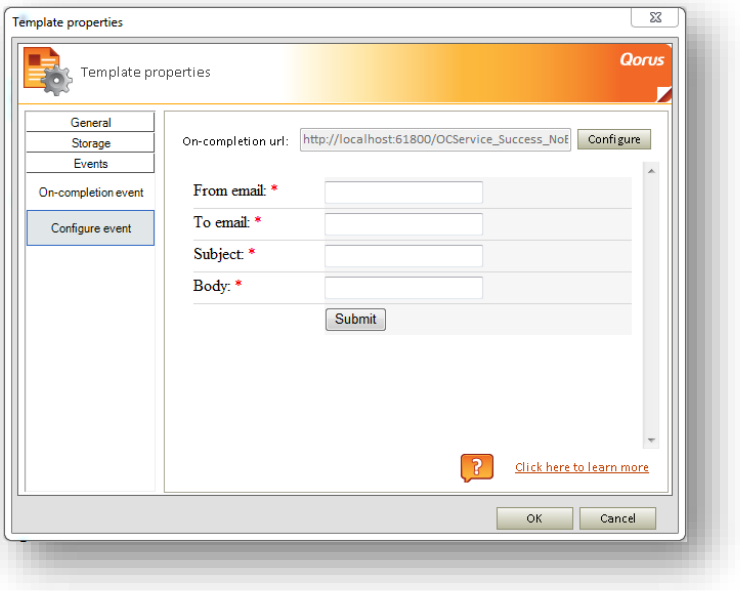


Figure 7: On-completion Event Screen with configuration page

Using the Qorus DocGeneration API

The Qorus DocGeneration API allows a developer to interact with DocGeneration programmatically. This can be done by either referencing the Qorus DocGeneration SharePoint API or by referencing the Qorus DocGeneration Merge Service API.

The Qorus DocGeneration SharePoint API is deployed to the SharePoint farm as part of the Qorus DocGeneration SharePoint installation.

The Qorus DocGeneration Merge Service API is installed as part of the Qorus DocGeneration Merge Service installation. This API can only be used if a merge server is setup. It is recommended to utilize the Merge Service API if you make use of Merge Servers as it will alleviate the processing load from the SharePoint Front End Servers. **Please note that only templates using User Defined Data Sources can be merged using this API.**

Qorus DocGeneration SharePoint API

To access the Qorus DocGeneration SharePoint API, create references to the following custom SharePoint web services:

"~/_layouts/15/QorusDocumentGeneration2013/QorusDocGen.asmx".
"~/_layouts/15/QorusSlideGeneration2013/QorusSlideGen.asmx"

If SharePoint 2010 is being used you are required to use the following URLs:

"~/_layouts/QorusDocumentGeneration/QorusDocGen.asmx".
"~/_layouts/QorusSlideGeneration/QorusSlideGen.asmx"

The web service can then be initialized as per the following code sample:

```
CODE SNIPPET FOR QORUS DOCGENERATION


QorusAPIService.QorusDocGen service = new QorusAPIService.QorusDocGen();
service.Credentials = CredentialCache.DefaultCredentials;
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

QorusAPIService.QorusSlideGen service = new QorusAPIService.QorusSlideGen();
service.Credentials = CredentialCache.DefaultCredentials;
```

Qorus DocGeneration Merge Service API

To access the Qorus DocGeneration Merge Service API, create a reference to the following WCF service: “http://mergeserver-webapp/QorusDocGen.svc”.

To access the Qorus SlideGeneration Merge Service API, create a reference to the following WCF service: “http://mergeserver-webapp/QorusSlideGen.svc”.

The WCF service can then be initialized as per the following code sample:

```
CODE SNIPPET FOR QORUS DOCGENERATION

public void CreateBindingAndEndPoint(string serviceURL, out System.ServiceModel.Channels.Binding
binding, out EndpointAddress endPointAddress)
{
   EndpointIdentity endpointIdentity = EndpointIdentity.CreateDnsIdentity("host/" +
serviceURL.Split('/')[2].Split(':')[0]);
   XmlDictionaryReaderQuotas quotas = new XmlDictionaryReaderQuotas();
   quotas.MaxArrayLength = int.MaxValue;
   quotas.MaxDepth = int.MaxValue;
   quotas.MaxStringContentLength = int.MaxValue;
   quotas.MaxNameTableCharCount = int.MaxValue;
   quotas.MaxBytesPerRead = int.MaxValue;
   WSHttpBinding WSHttpBinding = new WSHttpBinding();
   WSHttpBinding.ReaderQuotas = quotas;
   WSHttpBinding.ReliableSession.Enabled = true;
   WSHttpBinding.ReliableSession.Ordered = true;
   WSHttpBinding.ReliableSession.InactivityTimeout = new TimeSpan(0, 10, 0);
   WSHttpBinding.MaxReceivedMessageSize = int.MaxValue;
   WSHttpBinding.MaxBufferPoolSize = int.MaxValue;
   WSHttpBinding.CloseTimeout = new TimeSpan(0, 1, 0, 0);
   WSHttpBinding.OpenTimeout = new TimeSpan(0, 1, 0, 0);
   WSHttpBinding.SendTimeout = new TimeSpan(0, 1, 0, 0);
   WSHttpBinding.ReceiveTimeout = new TimeSpan(0, 1, 0, 0);

   AddressHeader addressHeader1 = AddressHeader.CreateAddressHeader("specialservice1",
serviceURL.Substring(0, serviceURL.LastIndexOf("/")), 1);
   AddressHeader[] addressHeaders = new AddressHeader[1] { addressHeader1 };
   EndpointAddress endPoint = new System.ServiceModel.EndpointAddress(new Uri(serviceURL),
endpointIdentity, addressHeaders);
   binding = WSHttpBinding;
   endPointAddress = endPoint;
}

public void Initialize()
{
    ChannelFactory<DocGenWcf.IQorusDocGen> templateFactory = null;

    System.ServiceModel.Channels.Binding binding;
    EndpointAddress endPointAddress;

    CreateBindingAndEndPoint("http://mergeserver-webapp/qorusdocgen.svc", out binding, out
endPointAddress);
    templateFactory = new ChannelFactory<DocGenWcf.IQorusDocGen>(binding, endPointAddress);

    foreach (OperationDescription op in templateFactory.Endpoint.Contract.Operations)
    {
       DataContractSerializerOperationBehavior dataContractBehavior =
op.Behaviors.Find<DataContractSerializerOperationBehavior>() as
DataContractSerializerOperationBehavior;
       if (dataContractBehavior != null)
       {
           dataContractBehavior.MaxItemsInObjectGraph = int.MaxValue;
       }
    }

    DocGenWcf.IQorusDocGen service = templateFactory.CreateChannel();
}
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

public void Initialize()
{
    ChannelFactory<SlideGenWcf.IQorusSlideGen> templateFactory = null;

    System.ServiceModel.Channels.Binding binding;
    EndpointAddress endPointAddress;

    CreateBindingAndEndPoint("http://mergeserver-webapp/qorusslidegen.svc", out binding, out
endPointAddress);
    templateFactory = new ChannelFactory<SlideGenWcf.IQorusSlideGen>(binding, endPointAddress);

    foreach (OperationDescription op in templateFactory.Endpoint.Contract.Operations)
    {
        DataContractSerializerOperationBehavior dataContractBehavior =
op.Behaviors.Find<DataContractSerializerOperationBehavior>() as
DataContractSerializerOperationBehavior;
        if (dataContractBehavior != null)
        {
            dataContractBehavior.MaxItemsInObjectGraph = int.MaxValue;
        }
    }

    SlideGenWcf.IQorusSlideGen service = templateFactory.CreateChannel();
}
```

The following is an overview of key methods that are exposed via the API services:

- CreateTemplate allows the creation of a Qorus Generation template via the API. The method can create a new blank template or be used to update an existing document to be a template.
- CreatePublishedTemplate will perform the same action as the CreateTemplate method mentioned above but will also publish the template once it is created.
- CreateTemplateAdvanced allows the developer to create a template by specifying a template creation object which describes how the template should be created. This object can be retrieved by making use of the RetrieveTemplateObject method.
- GetTemplateAssociation will return the GUID of the SharePoint list that a template is associated to. Returns NULL if no association exists.
- ProcessQorusTemplate allows the developer to merge a Qorus template.
- ProcessQorusTemplateNoReturn allows the developer to merge a Qorus template and not return the resulting byte array. This method is only available for DocGeneration.
- ProcessQorusTemplateByAssociatedItemID can be used to initiate a template merge by supplying the item ID of an item in the SharePoint list that the template is associated to. DocGeneration will extract the input values based on this ID and perform the merge.
- ProcessQorusTemplateUsingXmlData performs a merge but instead of accepting the standard array of input tags it accepts a XML file. For a complete discussion of this method and the layout of the file please refer to the “Creation of an XML data file for use with API merging” section.
- RefreshMergedDocument refreshes the data of a template, now represented as a byte array.

- RefreshMergedSharePointDocument refreshes the data of a template that is now stored in a SharePoint document library.
- RestitchDocument allows the developer to change and/or refresh the data of a merged document (one that has already been created by a merge or a stitch of multiple Qorus templates). Even a merged document that has been refreshed (through the RefreshMergedDocument or RefreshMergedSharePointDocument methods) can be restitched. This method allows the developer to make changes to the merged document without losing any extra added text that has already been added to the merged document and then saved. Examples of changes that the developer might like to make are: change the order of certain stitches (in the case of multiple embedded documents/Qorus templates), change the values of certain input fields, remove certain stitches (in the case of multiple embedded documents/Qorus templates), embed extra documents/Qorus templates into the merged document (add more stitches to the merged document). This method is only available for DocGeneration.
- RetrieveEntireQorusTemplateInputTagsSet retrieves a list of input tags required by a Qorus template as well as the fields that are associated to the list linked to the template.
- RetrieveMergedDocumentInputTags retrieves the input tags that were used to a merge a document, now represented as a byte array.
- RetrieveMergedDocumentStitches retrieves all the stitch objects that were originally stitched into the merged document as a byte array. It is needed for the restitching of merged documents. This method is only available for DocGeneration.
- RetrieveMergedSharePointDocumentInputTags retrieves the input tags that were used to merge a DocGeneration document that is now stored in a SharePoint document library.
- RetrieveQorusTemplate retrieves a byte array representing the template document.
- RetrieveQorusTemplateInputTags retrieves a list of input tags required by a Qorus template.
- RetrieveQorusTemplateMappedFields retrieves a list of the fields that are mapped to tags. Retrieving input tags and mapped fields for a template
- RetrieveStitchObject retrieves a two dimensional object array which is used to apply your stitch properties.
- RetrieveTemplateObject retrieves an object array which is used to specify template creation parameters.
- RetrieveTemplatesBySite retrieves all DocGeneration templates that reside in the template gallery, for the specified site.
- RetrieveOnCompletionCatalogueConfigXml will retrieve configuration data that is saved against an on-completion service at a catalogue level.

- RetrieveOnCompletionConfigXml allows you to retrieve on-completion configuration data that has been saved at a template level. This is saved via the template properties dialog when creating the template.
- SaveOnCompletionCatalogueConfigXml is used to save configuration data for an on-completion service against a catalogue. This can be viewed in SharePoint Central Administration.
- StitchDocument allows the developer to embed multiple documents/Qorus templates into another single Qorus template.
- StitchDocumentAdvanced is similar to the 'StitchDocument' method but it allows the developer to specify whether stitch placeholders that receive no content should remain in the final merged document or should be removed.
- StitchDocumentByAssociatedItemID performs the same task as the 'StitchDocument' method but instead of an array of input parameters; it accepts the item ID of an item in the template's associated list. This item is then used to generate the input values.
- StitchDocumentWithOncompletionWebservice is the same as the 'StitchDocument' method however it allows the developer to specify an on-completion web service to call when execution is complete.
- SetTemplateUserDefinedDatasource provides the developer with the ability to create a user defined data source in a template programmatically. This is achieved by supplying the QDF file that defines the data source as an XML parameter. If a data source with the same name already exists, it will be replaced.
- SyncPublishedTemplateToMergeServer can be used to send the latest published version of a template to the merge server database. This method can be used if the merge server was offline and the template was not saved to the database when publishing occurred.

Retrieving input tags and mapped fields for a template

```
CODE SNIPPET FOR QORUS DOCGENERATION

string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.docx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

object[][] inputTags = service.RetrieveQorusTemplateInputTags(strTemplateURL, iLCID);
object[][] allInputTags = service.RetrieveEntireQorusTemplateInputTagsSet(strTemplateURL, iLCID);
object[][] mappedFields = service.RetrieveQorusTemplateMappedFields(strTemplateURL, iLCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.pptx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

object[][] inputTags = service.RetrieveQorusTemplateInputTags(strTemplateURL, iLCID);
object[][] allInputTags = service.RetrieveEntireQorusTemplateInputTagsSet(strTemplateURL, iLCID);
object[][] mappedFields = service.RetrieveQorusTemplateMappedFields(strTemplateURL, iLCID);
```

- The *RetrieveQorusTemplateInputTags* method requires the following two parameters:
  - A URL that points to the template you wish to retrieve.
  - A Windows locale identifier to identify the current culture.

- The method will return an array of object arrays. Three values are stored per entry:
  - The name of the tag.
  - The value assigned to the tag. This will be empty, unless the tag is assigned a user value in Qorus.
  - The data type associated with the tag. Note that the actual type must match is associated data type or an exception will be thrown.
  - Also note that the data type of the tag can also be a whole data table. This allows the developer to have tags and table tags (also with tags) as input tags of a table.

- The *RetrieveQorusTemplateMappedFields* method requires the following two parameters:
  - A URL that points to the template you wish to retrieve.
  - A Windows locale identifier to identify the current culture.

- The method will return an array of object arrays. Two values are stored per entry:
  - The name of the field that is mapped.
  - The data type of the field.

Retrieving all templates on a SharePoint site

```
CODE SNIPPET FOR QORUS DOCGENERATION

string siteURL = "http://servername:100/mysite"
string[][] templates = service.RetrieveTemplatesBySite (siteURL, iLCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string siteURL = "http://servername:100/mysite"
string[][] templates = service.RetrieveTemplatesBySite (siteURL, iLCID);
```

- The *RetrieveTemplatesBySite* method requires the following two parameters:
  - A URL that points to a SharePoint site.
  - A Windows locale identifier to identify the current culture.

- The method will return an array of string arrays. Three string values are stored per entry:
  - The name of the template (including an extension)
  - The full URL of the template (including an extension)
  - The template description.

Merging a Qorus template

```
CODE SNIPPET FOR QORUS DOCGENERATION

string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.docx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

byte[] templateDoc = service.RetrieveQorusTemplate(strTemplateURL, iLCID);

//This table will be a nested table contained in CountryCodeTable
DataTable _CurrencyTable = new DataTable("CurrencyTable");
_CurrencyTable.Columns.Add("Unit", System.Type.GetType("System.String"));
_CurrencyTable.Columns.Add("Value", System.Type.GetType("System.Int32"));
_CurrencyTable.Rows.Add("$", 8);
_CurrencyTable.Rows.Add("R", 1);

DataTable _CountryCodeTable = new DataTable("CountryCodeTable");
_CountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_CountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
//Nested tables are of type System.String, they are also serialised
_CountryCodeTable.Columns.Add("CurrencyTable", System.Type.GetType("System.String"));
_CountryCodeTable.Rows.Add("US", 1, SerialiseDataTable(_CurrencyTable));
_CountryCodeTable.Rows.Add("ZA", 2, SerialiseDataTable(_CurrencyTable));

object[][] mappedFields = new object [1][];
object [][] inputTags = new object [2][];

mappedFields[0] = new object [] { "ID", "LetterID" };

inputTags[0] = new object [] { "LetterID", 2, "System.Int32" };
inputTags[1] = new object [] { "Country", "Germany", "System.String" };
inputTags[2] = new object [] { "Balance", Convert.ToDecimal(1000), "System.Decimal" };
inputTags[3] = new object [] { "CountryCodeTable", SerialiseDataTable(_CountryCodeTable),
"System.Data.DataTable" };

byte[] mergedDocument = service.ProcessQorusTemplate(templateDoc, strTemplateURL, inputTags,
                                                     mappedFields, iLCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.pptx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

byte[] templateDoc = service.RetrieveQorusTemplate(strTemplateURL, iLCID);

DataTable _CountryCodeTable = new DataTable("CountryCodeTable");
_CountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_CountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_CountryCodeTable.Rows.Add("US", 1);
_CountryCodeTable.Rows.Add("ZA", 2);

object[][] mappedFields = new object [1][];
object [][] inputTags = new object [2][];

mappedFields[0] = new object [] { "ID", "LetterID" };

inputTags[0] = new object [] { "LetterID", 2, "System.Int32" };
inputTags[1] = new object [] { "Country", "Germany", "System.String" };
inputTags[2] = new object [] { "Balance", Convert.ToDecimal(1000), "System.Decimal" };
inputTags[3] = new object [] { "CountryCodeTable", SerialiseDataTable(_CountryCodeTable),
"System.Data.DataTable" };

byte[] mergedDocument = service.ProcessQorusTemplate(templateDoc, strTemplateURL, inputTags,
                                                     mappedFields, iLCID);
```

- The *RetrieveQorusTemplate* method requires the following two parameters:
  - A URL that points to the template you wish to retrieve.
  - A Windows locale identifier to identify the current culture.
- The *ProcessQorusTemplate* accepts five parameters:
  - The template to be merged; represented as a byte array.
  - A URL that points to the template you wish to merge.
  - An array of object arrays that make up the input tags of the template. Three values are stored per entry:
    - The name of the tag.
    - The value assigned to the tag.
    - The data type associated with the tag. Note that the actual type must match is associated data type or an exception will be thrown.
- An array of object arrays that stores name/value pairs. If the template is set to write to a document library on merge, this array will indicate which tags are mapped to which fields in the destination list.
- **[SlideGeneration only]** A boolean that indicates whether to return the merged document (true) or not (false).
- A Windows locale identifier to identify the current culture.
- Once the merge has been completed, a byte array containing the merged document will be returned.
- When a template is merged via the API, it will not attach the merged document to any list items, even if it has been configured in the template properties to do so.

Merging a Qorus template without returning the resulting document [DocGeneration only]

The *RetrieveQorusTemplateNoReturn* method is used to merge a template without returning the results

```
CODE SNIPPET FOR QORUS DOCGENERATION

byte[] mergedDocument = service.ProcessQorusTemplateNoReturn(templateDoc, strTemplateURL, inputTags,
                                                    mappedFields, iLCID);
```

[SlideGeneration only]

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

byte[] mergedDocument = service.ProcessQorusTemplate(templateDoc, strTemplateURL, inputTags,
                                                    mappedFields, false, iLCID);
```

The *ProcessQorusTemplate* method accepts an additional boolean parameter. This indicates whether to return the merge document (true) or not (false).

Refreshing a Qorus template from a byte array

```
CODE SNIPPET FOR QORUS DOCGENERATION

string documentPath = "C:\\SamplePath\\SampleDocument.docx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

DataTable _NewCountryCodeTable = new DataTable("CountryCodeTable");
_NewCountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_NewCountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_NewCountryCodeTable.Rows.Add("GDB", 1);
_NewCountryCodeTable.Rows.Add("GM", 2);

byte[] document = File.ReadAllBytes(documentPath);

object[][] updatedInputTags = service.RetrieveMergedDocumentInputTags(document, cultureInfo.LCID);
updatedInputTags[0][1] = 3;
updatedInputTags[1][1] = "newValueForSecondTag";
updatedInputTags[2][1] = Convert.ToDecimal(2000);
updatedInputTags[3][1] = SerialiseDataTable(_NewCountryCodeTable);

byte[] refreshedDocument = service.RefreshMergedDocument(document, updatedInputTags, true,
cultureInfo.LCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string documentPath = "C:\\SamplePath\\SampleDocument.pptx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

DataTable _NewCountryCodeTable = new DataTable("CountryCodeTable");
_NewCountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_NewCountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_NewCountryCodeTable.Rows.Add("GDB", 1);
_NewCountryCodeTable.Rows.Add("GM", 2);

byte[] document = File.ReadAllBytes(documentPath);

object[][] updatedInputTags = service.RetrieveMergedDocumentInputTags(document, cultureInfo.LCID);
updatedInputTags[0][1] = 3;
updatedInputTags[1][1] = "newValueForSecondTag";
updatedInputTags[2][1] = Convert.ToDecimal(2000);
updatedInputTags[3][1] = SerialiseDataTable(_NewCountryCodeTable);

byte[] refreshedDocument = service.RefreshMergedDocument(document, updatedInputTags, true,
cultureInfo.LCID);
```

- The *RetrieveMergedDocumentInputTags* method requires the following two parameters:
  - A byte array that represents a merged Qorus document.
  - A Windows locale identifier to identify the current culture.
- The *RefreshMergedDocument* method accepts 4 parameters:
  - The document to be refreshed; represented as a byte array.
  - The input tags that were used to merge the template originally, these may contain updated values.
  - A boolean that indicates whether the template should be refresh base on the original template version (false), or the latest published version of the template (true).
  - A Windows locale identifier to identify the current culture.

Refreshing a Qorus template located in a SharePoint document library

```
CODE SNIPPET FOR QORUS DOCGENERATION

string documentURL = @"http://servername:100/Shared Documents/Merged Proposal Document.docx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

DataTable _NewCountryCodeTable = new DataTable("CountryCodeTable");
_NewCountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_NewCountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_NewCountryCodeTable.Rows.Add("GDB", 1);
_NewCountryCodeTable.Rows.Add("GM", 2);

object[][] updatedInputTags = service.RetrieveMergedSharePointDocumentInputTags(documentURL,
cultureInfo.LCID);
updatedInputTags[0][1] = 3;
updatedInputTags[1][1] = "newValueForSecondTag";
updatedInputTags[2][1] = Convert.ToDecimal(2000);
updatedInputTags[3][1] = SerialiseDataTable(_NewCountryCodeTable);

byte[] refreshedDocument = service.RefreshMergedSharePointDocument(documentURL, updatedInputTags,
true, true, cultureInfo.LCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string documentURL = @"http://servername:100/Shared Documents/Merged Proposal Document.pptx";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

DataTable _NewCountryCodeTable = new DataTable("CountryCodeTable");
_NewCountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_NewCountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_NewCountryCodeTable.Rows.Add("GDB", 1);
_NewCountryCodeTable.Rows.Add("GM", 2);

object[][] updatedInputTags = service.RetrieveMergedSharePointDocumentInputTags(documentURL,
cultureInfo.LCID);
updatedInputTags[0][1] = 3;
updatedInputTags[1][1] = "newValueForSecondTag";
updatedInputTags[2][1] = Convert.ToDecimal(2000);
updatedInputTags[3][1] = SerialiseDataTable(_NewCountryCodeTable);

byte[] refreshedDocument = service.RefreshMergedSharePointDocument(documentURL, updatedInputTags,
true, true, cultureInfo.LCID);
```

- The *RetrieveMergedSharePointDocumentInputTags* method requires the following two parameters:

  - A URL that points to a merged Qorus document.
  - A Windows locale identifier to identify the current culture.

- The *RefreshMergedSharePointDocument* method accepts 5 parameters:

  - A URL that points to the document you wish to refresh.
  - The input tags that were used to merge the template originally, these may contain updated values.
  - A boolean that indicates whether the template should be refresh base on the original template version (false), or the latest published version of the template (true).
  - A boolean that indicates whether the SharePoint document being refreshed, should automatically be overwritten with the resulting refreshed document.
  - A Windows locale identifier to identify the current culture.

Stitching a document

```
CODE SNIPPET FOR QORUS DOCGENERATION

int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;
string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.docx";
object[][] inputTags = service.RetrieveQorusTemplateInputTags(strTemplateURL, iLCID);

// stitch[0] = Url
// stitch[1] = Document
// stitch[2] = InputTags
// stitch[3] = Refresh
// stitch[4] = AppendIfStitchNotFound
// stitch[5] = IsQorusTemplate
// stitch[6] = LanguageId
// stitch[7] = StitchName
// stitch[8] = UsePublishedVersionOnRefresh
// stitch[9] = CarriageReturnBefore
// stitch[10] = CarriageReturnAfter
// stitch[11] = ExtraTextField
// stitch[12] = StitchID

DataTable _CountryCodeTable = new DataTable("CountryCodeTable");
_CountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_CountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_CountryCodeTable.Rows.Add("GDB", 1);
_CountryCodeTable.Rows.Add("GM", 2);

object[][] templateStitch = service.RetrieveStitchObject();
templateStitch[0][1] = "http://servername:100/QorusTemplates/TermsConditions.docx";
templateStitch[1][1] = null;
object[][] FilledStitchInputFields =
service.RetrieveQorusTemplateInputTags(templateStitch[0][1].ToString(), iLCID);
FilledStitchInputFields[0][1] = 2;
FilledStitchInputFields[1][1] = "String1";
FilledStitchInputFields[2][1] = Convert.ToDecimal(2000);
FilledStitchInputFields[3][1] = SerialiseDataTable(_CountryCodeTable);
templateStitch[2][1] = FilledStitchInputFields;
templateStitch[3][1] = true;
templateStitch[4][1] = true;
templateStitch[5][1] = true;
templateStitch[6][1] = iLCID;
templateStitch[7][1] = "Terms And Conditions";
templateStitch[8][1] = true;
templateStitch[9][1] = true;
templateStitch[10][1] = true;
templateStitch[11][1] = "Terms And Conditions Stitch";

object[][] documentStitch = service.RetrieveStitchObject();
documentStitch[0][1] = null;
documentStitch[1][1] = File.ReadAllBytes(@"C:\CompanyName\Documents\Proposal.docx");
documentStitch[2][1] = null;
documentStitch[3][1] = false;
documentStitch[4][1] = true;
documentStitch[5][1] = false;
documentStitch[6][1] = iLCID;
documentStitch[7][1] = "Proposal";
documentStitch[8][1] = null;
documentStitch[9][1] = true;
documentStitch[10][1] = true;
documentStitch[11][1] = "Proposal Stitch";

object[][][] stitches = new object[][][] { templateStitch, documentStitch };
service.StitchDocument(strTemplateURL, inputTags, stitches, iLCID);

byte[] stitchedDocument = service.StitchDocument(strTemplateURL, inputTags, stitches, iLCID);

// stitching using oncompletion webservice
object[][] tags = new object[][] { };
tags[0] = new object[] { "OutputPath", "//svr-09/output/products ", "System.String" };

string onCompletionWSUrl = "http://svr-09/webservice/ProductOnCompletion.asmx";
service.StitchDocumentWithOncompletionWebservice(strTemplateURL, inputTags, tags, stitches,
onCompletionWSUrl, iLCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;
string strTemplateURL = "http://servername:100/QorusTemplates/ProposalLetter.pptx";
object[][] inputTags = service.RetrieveQorusTemplateInputTags(strTemplateURL, iLCID);

// stitch[0] = Url
// stitch[1] = Document
// stitch[2] = InputTags
// stitch[3] = Refresh
// stitch[4] = AppendIfStitchNotFound
// stitch[5] = IsQorusTemplate
// stitch[6] = LanguageId
// stitch[7] = StitchName
// stitch[8] = UsePublishedVersionOnRefresh

DataTable _CountryCodeTable = new DataTable("CountryCodeTable");
_CountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_CountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_CountryCodeTable.Rows.Add("GDB", 1);
_CountryCodeTable.Rows.Add("GM", 2);

object[][] templateStitch = service.RetrieveStitchObject();
templateStitch[0][1] = "http://servername:100/QorusTemplates/TermsConditions.pptx";
templateStitch[1][1] = null;
object[][] FilledStitchInputFields =
service.RetrieveQorusTemplateInputTags(templateStitch[0][1].ToString(), iLCID);
FilledStitchInputFields[0][1] = 2;
FilledStitchInputFields[1][1] = "String1";
FilledStitchInputFields[2][1] = Convert.ToDecimal(2000);
FilledStitchInputFields[3][1] = SerialiseDataTable(_CountryCodeTable);
templateStitch[2][1] = FilledStitchInputFields;
templateStitch[3][1] = true;
templateStitch[4][1] = true;
templateStitch[5][1] = true;
templateStitch[6][1] = iLCID;
templateStitch[7][1] = "Terms And Conditions";
templateStitch[8][1] = true;

object[][] documentStitch = service.RetrieveStitchObject();
documentStitch[0][1] = null;
documentStitch[1][1] = File.ReadAllBytes(@"C:\CompanyName\Documents\Proposal.pptx");
documentStitch[2][1] = null;
documentStitch[3][1] = false;
documentStitch[4][1] = true;
documentStitch[5][1] = false;
documentStitch[6][1] = iLCID;
documentStitch[7][1] = "Proposal";
documentStitch[8][1] = null;

object[][][] stitches = new object[][][] { templateStitch, documentStitch };
service.StitchDocument(strTemplateURL, inputTags, stitches, iLCID);

byte[] stitchedDocument = service.StitchDocument(strTemplateURL, inputTags, stitches, iLCID);

// stitching using oncompletion webservice
object[][] tags = new object[][] { };
tags[0] = new object[] { "OutputPath", "//svr-09/output/products ", "System.String" };

string onCompletionWSUrl = "http://svr-09/webservice/ProductOnCompletion.asmx";
service.StitchDocumentWithOncompletionWebservice(strTemplateURL, inputTags, tags, stitches,
onCompletionWSUrl, iLCID);
```

- The *RetrieveStitchObject* method returns a two dimensional object array:
  - Each object array item is a single property for the stitch object.
  - There is a string object in the array that indicates what the property is.
    - *[0] Url* is used for documents that are to be retrieved from the web.
    - *[1] Document* is used for documents that are too passed through as a byte array

- *[2] InputTags* is only used when your stitch is a template. Use the *RetrieveQorusTemplateInputTags* method to get the correct input tags. This allows you to assign values to the input tags before the stitch is done.
- *[3] Refresh* allows the stitch object to be refreshed when the merged document is refreshed.
- *[4] AppendIfStitchIsNotFound* allows the developer to stitch into a template even if the stitch placeholder is not found. This will insert the stitches at the end of the document in the order they are in the array.
- *[5] IsQorusTemplate* indicates whether the document you want to stitch is a Qorus template.
- *[6] LanguageId* indicates what language id you would like the stitched template to use.
- *[7] StitchName* indicates which placeholder the stitch must be inserted at. Use the Multiple dynamic content sections so create a stitch placeholder.
- *[8] UsePublishedVersionOnRefresh* indicates whether a template stitch should get the last published version of the template. If this is null it will use whatever value is specified when a main document is refreshed.
- *[9] CarriageReturnBefore* **[DocGeneration only]** defaults to false if it is not specified. If it is set to true it indicates that an extra empty line should be added above the place where the stitch gets inserted. This can be used as a place in the stitched MS Word Document where users can add text after the stitch.
- *[10] CarriageReturnAfter* **[DocGeneration only]** defaults to false if it is not specified. If it is set to true it indicates that an extra empty line should be added after the stitch. This can be used as a place in the stitched MS Word Document where users can add text after the stitch.
- *[11] ExtraTextField* **[DocGeneration only]** defaults to null if it is not specified. This can be used as an extra text field by the developer in which to save any extra information.
- *[12] StitchID* **[DocGeneration only]** is where a stitch ID is maintained. The developer should not make any changes tho this field. The StitchID is used for Restitching of documents.

- The *StitchDocument* method requires the following parameters:

  - A URL that points to the template you wish to merge.
  - A two dimensional array of objects for the input tags of the template.
  - An array of all the stitches that need to be stitched into the template.
  - A Windows locale identifier to identify the current culture.

- The *StitchDocument* method returns a byte array which contains the stitched document.
- The StitchDocumentWithOncompletionWebservice method has two more parameters than StitchDocument. They are:

  - An array of objects that are used for tags for the on completion web service (if required)

- A URL that points to the on completion web service that you want it to call.

Restitching a merged document from a byte array [DocGeneration only]

CODE SNIPPET

```
byte[] document = File.ReadAllBytes(@"C:\\SamplePath\\SampleDocument.docx");
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

object[][] inputTags = service.RetrieveMergedDocumentInputTags(document, iLCID);

// get all the stitchobjects that exist inside the merged document, cast to a multidimentinal object
// and then cast to a list
byte[] stitchesBytes = ts.RetrieveMergedDocumentStitches(document, iLCID);
object[][][] stitchesArray = DeserialiaeArrayOfStringArray<object[][][]>(stitchesBytes);
List<object[][]> updatedStitches = stitchesArray.ToList();

// change the first cell of the first row of a table tag of the first stitch (at index 0)
// inside the merged document.
object[][] stitch0 = updatedStitches[0];
object[][] stitch0InputTags = (object[][])stitch0[2][1]; // stitch[2] = InputTags
string tableStr0 = stitch0InputTags[2][1].ToString();
DataTable table0 = DeserialiseDataTable(tableStr0);
table0.Rows[0][0] = "New Value";
tableStr0 = SerialiseDataTable(table0);
stitch0InputTags[2][1] = tableStr0; // stitch[2] = InputTags

// assign a whole new table to the table tag of the second stitch inside the merged document.
object[][] stitch1 = updatedStitches[1];
string URLStr1 = stitch1URL = (string)stitch1[0][1]; // stitch[0] = Url
object[][] FilledStitchInputFields1 = ts.RetrieveQorusTemplateInputTags(URLStr1, iLCID);
DataTable _NewCountryCodeTable = new DataTable("CountryCodeTable");
_NewCountryCodeTable.Columns.Add("Code", System.Type.GetType("System.String"));
_NewCountryCodeTable.Columns.Add("CodeID", System.Type.GetType("System.Int32"));
_NewCountryCodeTable.Rows.Add("GDB", 1);
_NewCountryCodeTable.Rows.Add("GM", 2);
FilledStitchInputFields1[0][1] = 2;
FilledStitchInputFields1[1][1] = "String1";
FilledStitchInputFields1[2][1] = Convert.ToDecimal(2000);
FilledStitchInputFields1[3][1] = SerialiseDataTable(_NewCountryCodeTable);
updatedStitches[1][2][1] = FilledStitchInputFields1; // stitch[2] = InputTags

// add another carriage return before the third stitch inside the merged document.
updatedStitches[2][9][1] = true; // stitch[10] = CarriageReturnBefore

// insert a new stitch at index 1 (It will become the new 2nd stitch of the document.
// It gets inserted right after the end of the 1st stitch inside the merged document)
object[][] newstitchToAdd9 = ts.RetrieveStitchObject();
string url9 = "http://comettest2010:300/sites/Sub/QorusTemplates/NewStitch1.docx";
object[][] FilledStitchInputFields9 = ts.RetrieveQorusTemplateInputTags(url9, iLCID);
FilledStitchInputFields9[0][1] = 3;
FilledStitchInputFields9[1][1] = "String3";
newstitchToAdd9[0][1] = url9;
newstitchToAdd9[2][1] = FilledStitchInputFields9; // InputTags
newstitchToAdd9[3][1] = true;                     // Refresh
newstitchToAdd9[4][1] = true;                     // AppendStitchIfNotFound
newstitchToAdd9[5][1] = true;                     // IsQorusTemplate
newstitchToAdd9[6][1] = iLCID;                    // LanguageID
newstitchToAdd9[7][1] = "APIDocStitch";           // StitchName
newstitchToAdd9[8][1] = true;                     // UsePublishedVersionOnRefresh
newstitchToAdd9[9][1] = true;                     // CarriageReturnBefore
newstitchToAdd9[10][1] = true;                    // CarriageReturnAfter
newstitchToAdd9[11][1] = "NEW STITCH 1";          // ExtraTextField
updatedStitches.Insert(1, newstitchToAdd9); // will insert new stitch at index 1

// delete the fourth stitch (at index 3)
updatedStitches.RemoveAt(3);

// add a new stitch to the end of the merged document
object[][] newstitchToAdd = ts.RetrieveStitchObject();
newstitchToAdd[0][1] = http://comettest2010:300/sites/Sub/QorusTemplates/NewStitch2.docx;
newstitchToAdd[2][1] = FilledStitchInputFields9; // InputTags
newstitchToAdd[3][1] = true;                     // Refresh
newstitchToAdd[4][1] = true;                     // AppendStitchIfNotFound
```

```
newstitchToAdd[5][1] = true;                    // IsQorusTemplate
newstitchToAdd[6][1] = iLCID;                   // LanguageID
newstitchToAdd[7][1] = "APIDocStitch";          // StitchName
newstitchToAdd[8][1] = true;                    // UsePublishedVersionOnRefresh
newstitchToAdd[9][1] = true;                    // CarriageReturnBefore
newstitchToAdd[10][1] = true;                   // CarriageReturnAfter
newstitchToAdd[11][1] = "NEW STITCH 2";         // ExtraTextField
updatedStitches.Add(newstitchToAdd);

//Now Write
bool usePublishedVersion = true;
stitchesBytes = SerialiseToByteArray(updatedStitches.ToArray());
byte[] reStichedDocument = ts.RestitchDocument(document, stitchesBytes, usePublishedVersion, iLCID);
System.IO.File.WriteAllBytes(@"C:\Temp\DocGenReStitch.docx", reStichedDocument);
```

- *RetrieveMergedDocumentStitches* retrieves all the stitch objects that were originally stitched into the merged document as a byte array.
- After that these stitch objects need to be cast into a multidimensional object by using the *DeserialiseArrayOfStringArray* method.
  - Here is an example piece of code that accepts a byte array, and then deserialises it to a multidimentional object.

```
CODE SNIPPET

public static T DeserialiseByteArray<T>(byte[] obj)
{
        if (obj == null)
        {
                return default(T);
        }
        using (MemoryStream MemoryStream = new MemoryStream())
        {
                 MemoryStream.Write(obj, 0, obj.Length);
                 MemoryStream.Seek(0, 0);
                 BinaryFormatter BinaryFormatter = new BinaryFormatter();
                 return (T)BinaryFormatter.Deserialise(MemoryStream);
            }
        }
```

- This multidimensional object then needs to be cast into a list (of multi-dimensional object arrays) to allow the developer insert and remove stitches at specific indexes. If `updatedStitches` is the list of all the stitch objects, `updatedStitches[0]` will be the stitch at index 0 in the document. It will be the first stitch in the document. Likewise `updatedStitches[1]` will be the stitch at index 1 in the document. It will be the second stitch in the document.
- Important note: In order to be able cast the multidimensional object into a list, the System.Linq object needs to be referenced in the project (`using System.Linq`)
- The developer can now edit the properties of a stitch object that exists at specific indexes in the merged document. If a table tag was part of the input fields of a stitch object and the developer would like to change the value of one of the cells in the table, he/she will need to deserialise the byte arrays that were sent through at the *InputTags* property of the stitch object into a data table first. This can be done with the De*serialiseDataTable* method below. After he/she changed the value at one of the cells, he/she will need to serialise the datatable back into a byte array again, before

sending it through to the *InputTags* property of the stitch object. The example piece of code (Figure 2: Serialise Data Table) that was given earlier can be re-used for this.

- Here is an example piece of code that accepts a byte array and then deserialises it to a DataTable type.

```
CODE SNIPPET

 private string DeserialiseDataTable(string tableXmlStr)
  {
       if (String.IsNullOrEmpty(tableXmlStr))
       {
              return new DataTable();
       }
       using (StringReader reader = new StringReader(tableXmlStr))
       {
              DataTable dataTable = new DataTable();
              System.Xml.XmlReader xmlReader = System.Xml.XmlReader.Create(reader);
              dataTable.ReadXml(xmlReader);
              return dataTable;
       }
  }
```

- To remove a stitch at a specific index, the *RemoveAt* method can be used. It accepts the index of the stitch that needs to be removed as a parameter.
- To insert a stitch at a specific index, the *Insert* method can be used. It accepts the index of the place where the stitch needs to be inserted as well as a multi-dimensional object array (the new stitchobject) as parameters. If the index provided was 2, it means that the new stitch will be inserted directly below the stitchobject that is at index 1. Take note that this also implies the following: If any text that is not part of a stitch object exists below the stitch object at index 1 and a new stitch object then gets inserted at index 2, the text will be moved to after the new stitch. E.g. If the document consisted of stitchobject [0], stitchobject [1], normal text, stitchobject[2], the restitched document will consist of stitchobject[0], stitchobject[1], Newstitchobject[2], normal text, stitchobject[3(was at 2)].
- To add a new stitch to the end of the merged document, the *Add* method can be used. It accepts a multi-dimensional object array (the new stitchobject) as a parameter.
- Before the restitch, the list of multidimensional object arrays needs to be serialised to a byte array again. Use the following code example to serialise the array.
- Here is an example piece of code that accepts any type, and serialises it to a byte array again.

```
CODE SNIPPET

public static byte[] SerializeToByteArray<T>(T obj)
{
        if (obj == null)
        {
                return null;
        }
        using (MemoryStream MemoryStream = new MemoryStream())
        {
                 System.Runtime.Serialization.Formatters.Binary.BinaryFormatter BinaryFormatter = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
                 BinaryFormatter.Serialize(MemoryStream, obj);
                 MemoryStream.Seek(0, 0);
                 return MemoryStream.ToArray();
            }
        }
```

- The *RestitchDocument* method finally restitches the merged document, so that it inserts all stitch object with their updated information; as well as any extra text that exists inside the already merged document, but is not part of a stitch object. It accepts 4 parameters:

  - A byte array that is the original merged document.
  - A byte array that is the stitch objects.
  - A boolean that indicates whether the template should be refresh base on the original template version (false), or the latest published version of the template (true).

Creating a template

```
CODE SNIPPET FOR QORUS DOCGENERATION

string documentLibraryURL = @"http://servername:100/Shared Documents";
string baseDocumentPath = @"C:\\my documents\TemplateBase.docx";
string templateName = "MyTemplate";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

byte[] baseDocument = File.ReadAllBytes(baseDocumentPath);

service.CreateTemplate(documentLibraryURL, templateName, baseDocument, iLCID);
```

```
CODE SNIPPET FOR QORUS SLIDEGENERATION

string documentLibraryURL = @"http://servername:100/Shared Documents";
string baseDocumentPath = @"C:\\my documents\TemplateBase.pptx";
string templateName = "MyTemplate";
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

byte[] baseDocument = File.ReadAllBytes(baseDocumentPath);

service.CreateTemplate(documentLibraryURL, templateName, baseDocument, iLCID);
```

- The *CreateTemplate* method accepts 4 parameters:

  - A URL to a document library. The template will be created in this library. You can include the folder in the path if you wish to create the template in a folder within the library.
    If you specify the path and name of a document that already exists, the document

will be converted to a template. Any content type metadata that is stored against the document item will be maintained.
- The name of the template. (Without extension)
- A byte array of an existing document/presentation to base the template's content on. [Optional] Can be set to NULL.
- A Windows locale identifier to identify the current culture.

Programmatically adding a user defined data source to a template

```
CODE SNIPPET FOR QORUS DOCGENERATION

string templateUrl = @"http://servername:100/QorusTemplates/Sample.docx";
string dataSourceXml = System.IO.File.ReadAllText(@"C:\\sources\MyDataSource.qdf");
int iLCID = System.Globalization.CultureInfo.CurrentCulture.LCID;

service.SetTemplateUserDefinedDatasource(templateUrl, dataSourceXml, iLCID);
```

- The *SetTemplateUserDefinedDataSource* method requires the following three parameters:

  - A URL that points to the DocGeneration template for which you would like to add the data source.
  - A string that contains the XML that defines a QDF file. This can be generated by creating a user defined data source in the DocGeneration add-in and then selecting the “Export” option. This will save a QDF file which you can then supply to this method.
  - A Windows locale identifier to identify the current culture.

Creation of QDF Files to Maintain User-Defined Data Sources that contain Nested Tables

User-defined data sources that contain nested tables can only be created from the API. (With nested we refer to tables that exist inside other tables.)

The Qorus Generation framework allows a developer to build a qdf file that provides the information required to create a data source that contains nested tables.

These qdf files are then imported at the front end into the DocGen Qorus template or SlideGen Qorus template to create the user-defined data source that contains the nested tables. Please refer to the Training Manual Qorus DocGeneration or the Training Manual Qorus SlideGeneration Suite for more details on Importing a User-Defined Data Source.

The way to structure the code for creating a user-defined data source with a nested table will be illustrated with the following example:

User Defined Data Source with a Table and a Nested Table

The user-defined data source of this example consists of the following:

- A table, 'MasterTable', that contains a data tag, InvoiceNumber; as well as a nested table tag: 'InvoiceTable'.
- The nested table, 'InvoiceTable', in turn contains 6 data tags.

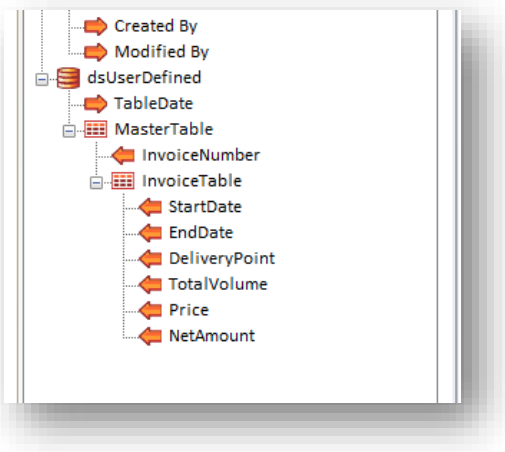


Figure 8: User Defined Data Source with a Table and a Nested Table

The code snippet to create this data source with is the following:

```
CODE SNIPPET

<DataSource Name="dsUserDef">
  <Tags>
   <Tag Name="MasterTable" FriendlyName="MasterTable" Type="System.Data.DataTable" IsTable="True"
IsInputTag="False" IsImageTag="False">
      <ChildTag Name="InvoiceNumber" FriendlyName="InvoiceNumber" Type="System.String"
IsInputTag="False" IsTable="False" IsImageTag="False" />
      <ChildTag Name="InvoiceTable" FriendlyName="InvoiceTable" Type="System.String"
IsInputTag="False" IsTable="True" IsImageTag="False">
        <ChildTag Name="StartDate" FriendlyName="StartDate" Type="System.String" IsInputTag="False"
IsTable="False" IsImageTag="False" />
        <ChildTag Name="EndDate" FriendlyName="EndDate" Type="System.String" IsInputTag="False"
IsTable="False" IsImageTag="False" />
        <ChildTag Name="DeliveryPoint" FriendlyName="DeliveryPoint" Type="System.String"
IsInputTag="False" IsTable="False" IsImageTag="False" />
       <ChildTag Name="TotalVolume" FriendlyName="TotalVolume" Type="System.String"
IsInputTag="False" IsTable="False" IsImageTag="False" />
        <ChildTag Name="Price" FriendlyName="Price" Type="System.String" IsInputTag="False"
IsTable="False" IsImageTag="False" />
        <ChildTag Name="NetAmount" FriendlyName="NetAmount" Type="System.String" IsInputTag="False"
IsTable="False" IsImageTag="False" />
      </ChildTag>
    </Tag>
  </Tags>
  <DataSourceMetadata Type="4">
  </DataSourceMetadata>
</DataSource>
```

- **DataSource Name** can be any name.
- Note that in the code above, for each of the two tables **IsTable="True"**
- Also note, the data type of the nested table is set to **Type="System.String"**

Creation of QTF Files to test User-Defined Data Sources with

The Qorus DocGeneration framework allows a developer to build qtf files that provide values for the user defined data tags and data table tags of User Defined Data Sources inside Qorus DocGeneration or Qorus SlideGeneration templates. These qtf files can then be used by users who do not have technical knowledge about the Qorus Generation API to test their template with.

The way to structure the code for assigning values to data tags and data table tags will be illustrated with two examples:

User Defined Data Source with Tags and a Table

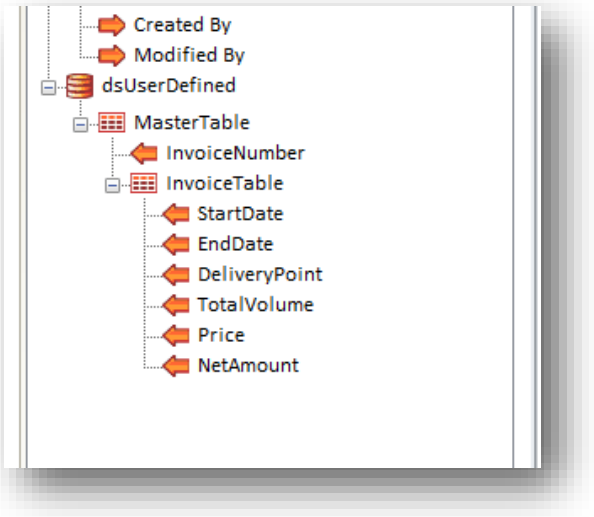


Figure 9: User Defined Data Source with Tags and a Table

- The code snippet below can be used to assign values to these tags.

```
CODE SNIPPET

<Data Version="1.0">
  <TableTag id="MasterTable" type="System.Data.DataTable">
   <Row>
      <Tag id="InvoiceNumber" type="System.String">7849344518</Tag>
      <TableTag id="InvoiceTable" type="System.Data.DataTable">
        <Row>
          <Tag id="StartDate" type="System.String">10 April 2012</Tag>
          <Tag id="EndDate" type="System.String">12 April 2012</Tag>
          <Tag id="DeliveryPoint" type="System.String">XYZ</Tag>
          <Tag id="TotalVolume" type="System.String">50</Tag>
          <Tag id="Price" type="System.String">34.00</Tag>
          <Tag id="NetAmount" type="System.String">30.00</Tag>
        </Row>
      </TableTag>
    </Row>
  </TableTag>
</Data>
```

- This code snipped needs to be saved as a .qtf file.
- The qtf file can then be used to test the template that contains this user defined data source as follows:

  - Open the template with the User Defined Data Source.
  - In the Qorus ribbon click ‘Test Template’

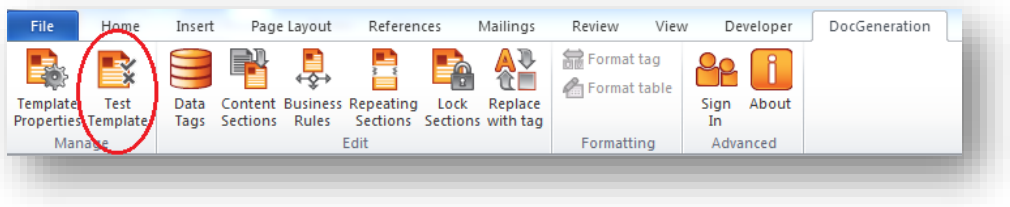


Figure 10: Click 'Test Template' in the Qorus ribbon of either the MS Word Document or the MS PowerPoint

- The Test Template dialog appears. Click ‘Select file’ and browse to the qtf file. After you have selected the qtf file, click OK to test the template.

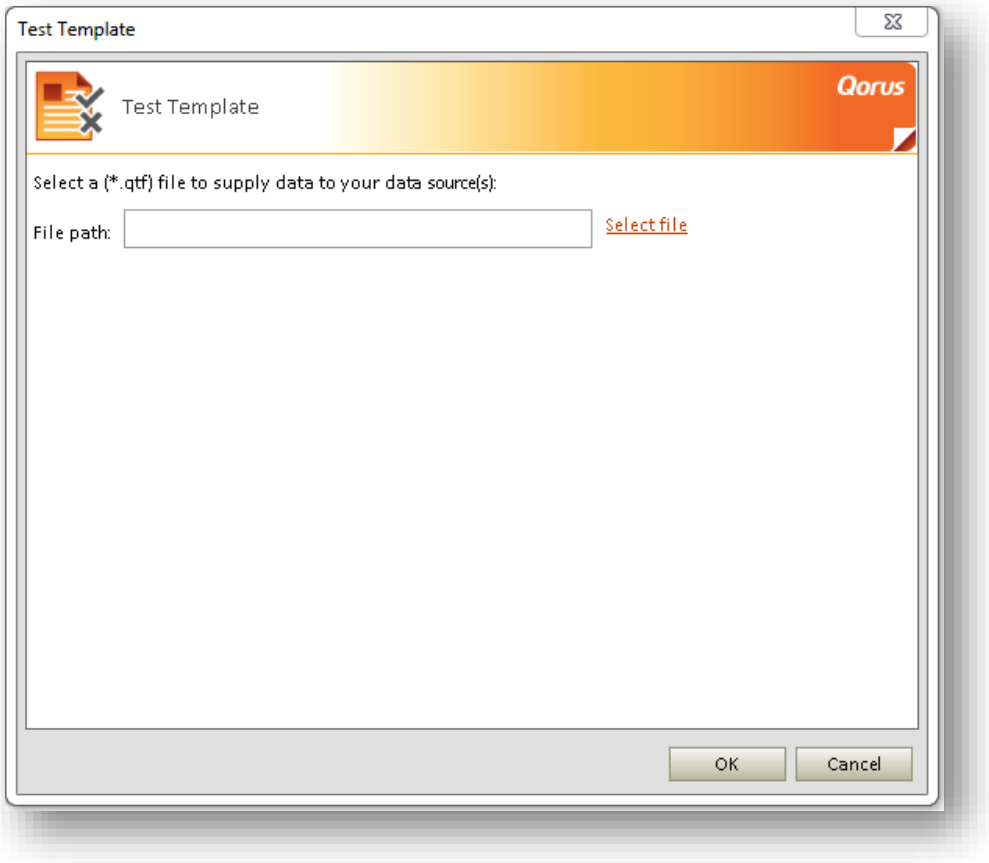
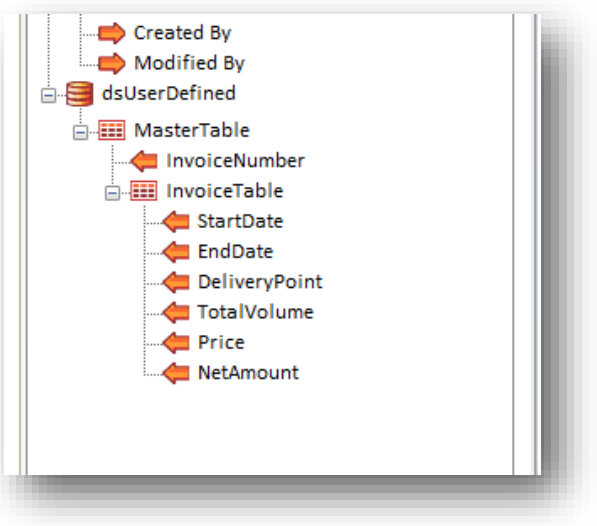


Figure 11: Click OK to test the template

- If more than one user defined data source exists in a template, the values of the tags and table tags from all the data sources can be within assigned within one qtf file.\

Creation of an XML data file for use with API merging

When performing Qorus Generation merges via the API, the framework allows the user to supply an XML data file, which will be used to populate the data tag values. This can be used instead of calling the "RetrieveQorusTemplateInputTags" method. The API method that accepts this file is called "ProcessQorusTemplateUsingXmlData". This file has a more technical layout than the standard QTF and the structure is illustrated in the below example.
This example makes use of the data source configured in the previous section:




```
CODE SNIPPET

<Data>
  <TableTag id="MasterTable" type="System.Data.DataTable">
   <Row>
      <Tag id="InvoiceNumber" type="System.String">7849344518</Tag>
      <TableTag id="InvoiceTable" type="System.Data.DataTable">
        <Row>
          <Tag id="StartDate" type="System.String">10 April 2012</Tag>
          <Tag id="EndDate" type="System.String">12 April 2012</Tag>
          <Tag id="DeliveryPoint" type="System.String">XYZ</Tag>
          <Tag id="TotalVolume" type="System.String">50</Tag>
          <Tag id="Price" type="System.String">34.00</Tag>
          <Tag id="NetAmount" type="System.String">30.00</Tag>
        </Row>
      </TableTag>
    </Row>
    <Row>
      <Tag id="InvoiceNumber" type="System.String">428563141T </Tag>
      <TableTag id="InvoiceTable" type="System.Data.DataTable">
        <Row>
          <Tag id="StartDate" type="System.String">29 April 2012</Tag>
          <Tag id="EndDate" type="System.String">30 April 2012</Tag>
          <Tag id="DeliveryPoint" type="System.String">ABC </Tag>
          <Tag id="TotalVolume" type="System.String">60</Tag>
          <Tag id="Price" type="System.String">42.00</Tag>
          <Tag id="NetAmount" type="System.String">40.00</Tag>
        </Row>
      </TableTag>
    </Row>
  </TableTag>
</Data>
```

Enabling custom list definitions for Qorus Generation

List templates supported natively

The following list templates are supported by Generation Suite natively:

- 100 – Generic List
- 101 – Document Library
- 104 – Announcements List
- 105 – Contacts List
- 107 – Tasks List
- 109 – Picture Library

If you wish to create your own list templates and activate Generation Suite products on these list templates, you will need to include the relevant custom actions.

Creating additional custom actions

There are two sets of custom actions available. One of these is for list templates that are based on the “Generic List” base type and the other for lists that are based on the “Document Library” base type.
Once you have created your list templates, you will be required to install these custom actions to associate the Registration ID of these list templates with the Qorus Generation suite custom actions.

Please contact the Qorus support center for assistance with regards to this matter.

If you have custom features deployed which references jQuery, you might experience script errors if the jQuery version conflicts with the version referenced by Qorus Generation.
To remedy this problem, navigate to the 14 hive folder, where the Qorus Generation features are deployed. You can then remove the jQuery reference (as illustrated below) from the MenuItems.xml feature files.

Resolving jQuery conflicts with customized sites

Please remember that you will have to re-register and activate the Qorus Generation features after making these changes.

```
<CustomAction ScriptSrc="QorusDocumentGeneration/jquery-1.7.2.min.js"
              Location="ScriptLink"
              Sequence="100">
</CustomAction>
```

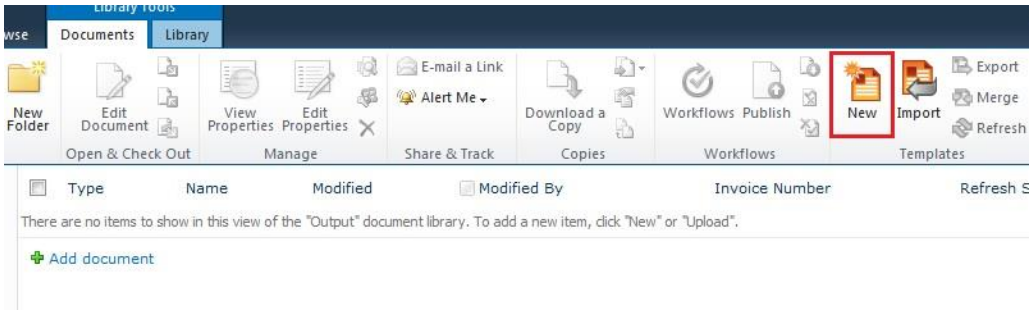
Example scenario

This will guide you through creating a simple template and merge it with a simple data source using the SharePoint API.

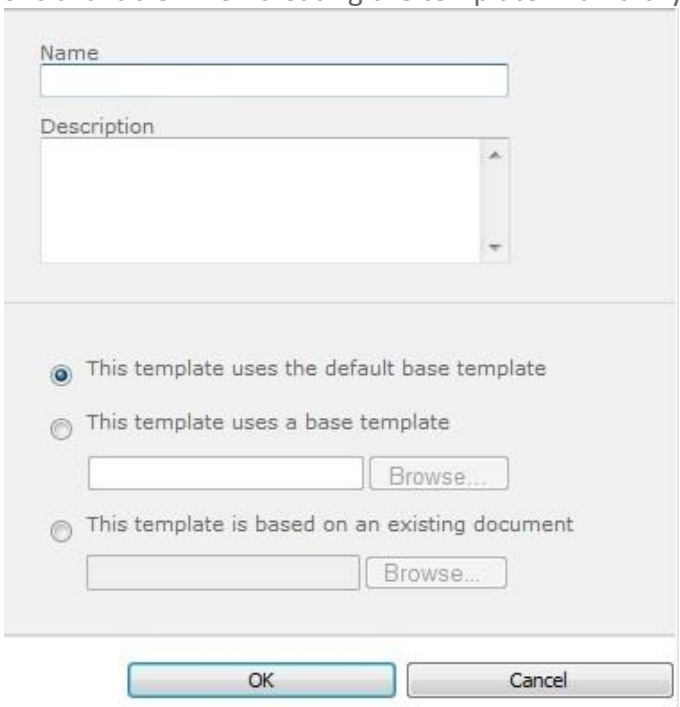
Create a template in a library

Qorus DocGeneration proposes two types of templates: templates that reside in the Qorus template gallery and that are associated to a list and templates that reside in a SharePoint library. For this example, we are using a template residing in a SharePoint library.

Create a SharePoint Document library and enable versioning on it. Create a new template as shown below.



There are three options available when creating the template in a library.



This template uses the default base template

This option denotes the template to be based on the default stored blank template on the machine that Qorus places in the file system.

This template uses a base template

This option allows the user to choose from templates that have been created on the SharePoint site via a template gallery and denoted at time of creation to be base templates. Note: The created base template will have to be published in order for it to be available in the list of base templates.

This template is based on an existing document

This option allows a .docx file on the machines file system to be used as a template to base the template to be created, on. This is essentially turning the word document into a template and placing it in the library.

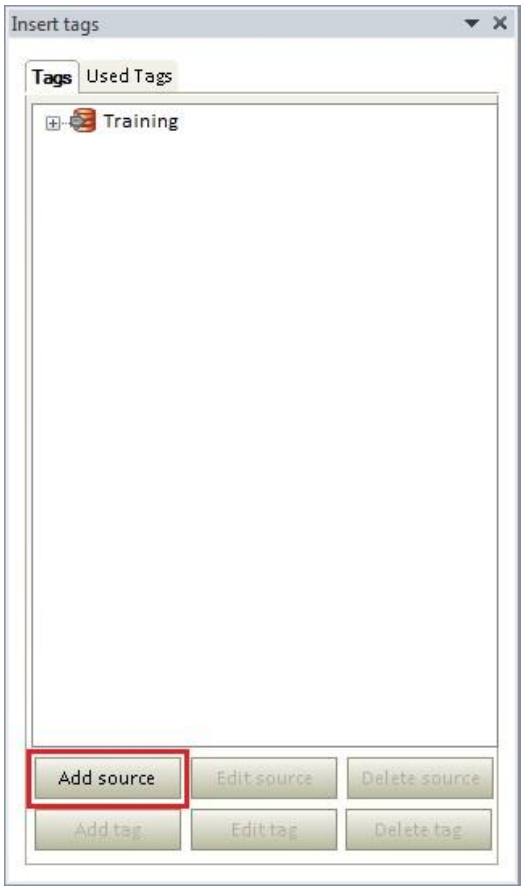
Please select the "This template uses the default base template" **option for our example.**

Create an example User Defined Data Source

Once your template is created, open it and create a data source. User-Defined Data Sources allow tags to be defined in the template, but their values only get set once the template gets merged using the Qorus API from a calling application written by a software developer. You can now create a data source and data tags that meet your need by following the steps below.

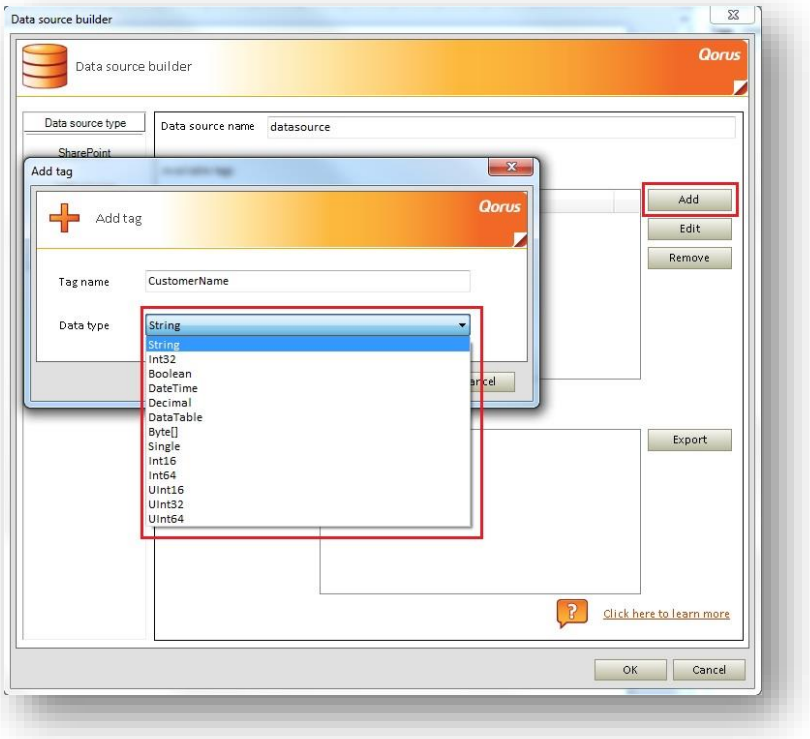
- Click on the Data Tags button in the Qorus ribbon.

- Click “Add source” to open the data source window:

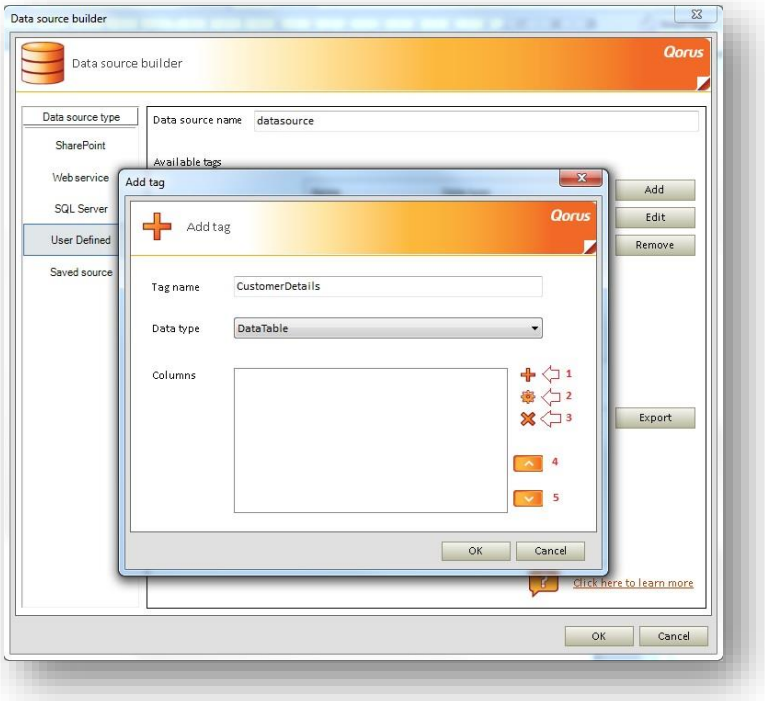


- Under Data Source Type on the left, ensure User Defined is selected.
- Enter a name in the Data Source name field.
- Now Data Tags can be added by clicking the “Add” button.

When it is clicked an Add Tag window appears to set properties of the Data Tag. A name must be given and the data type must be chosen.



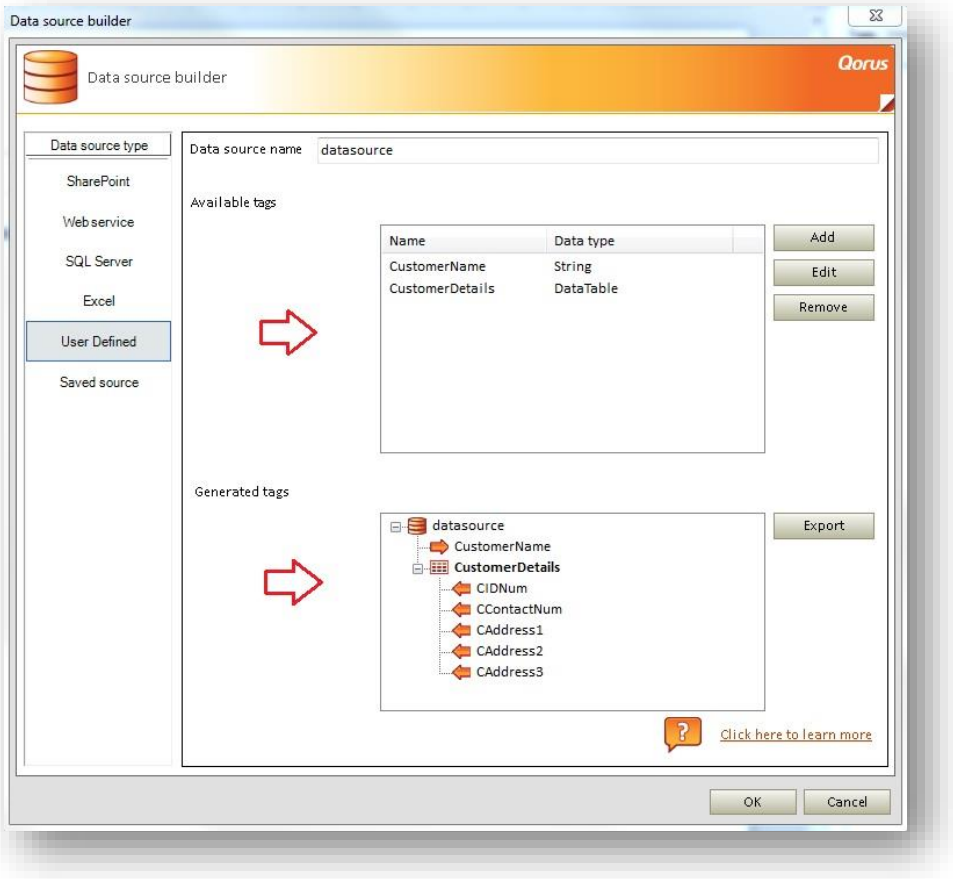
- A Table Tag can also be created from the “front end interface” by selecting the “DataTable” data type.



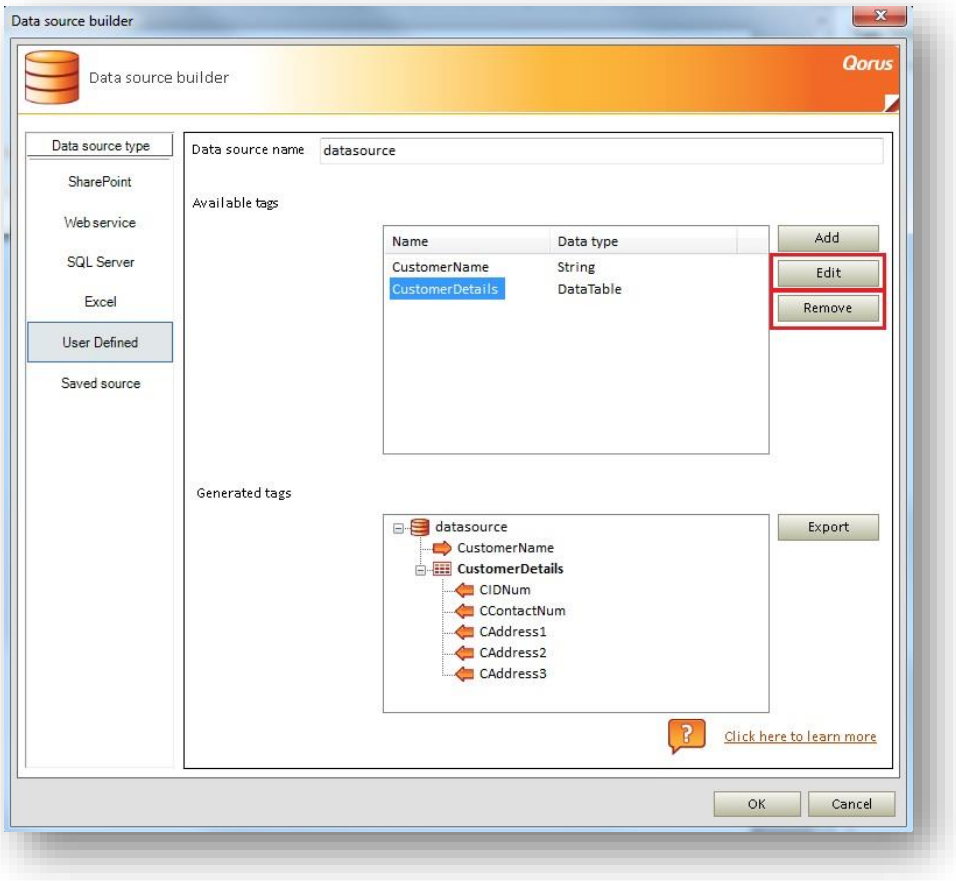
Buttons:

- Adds a Data Tag to the table.

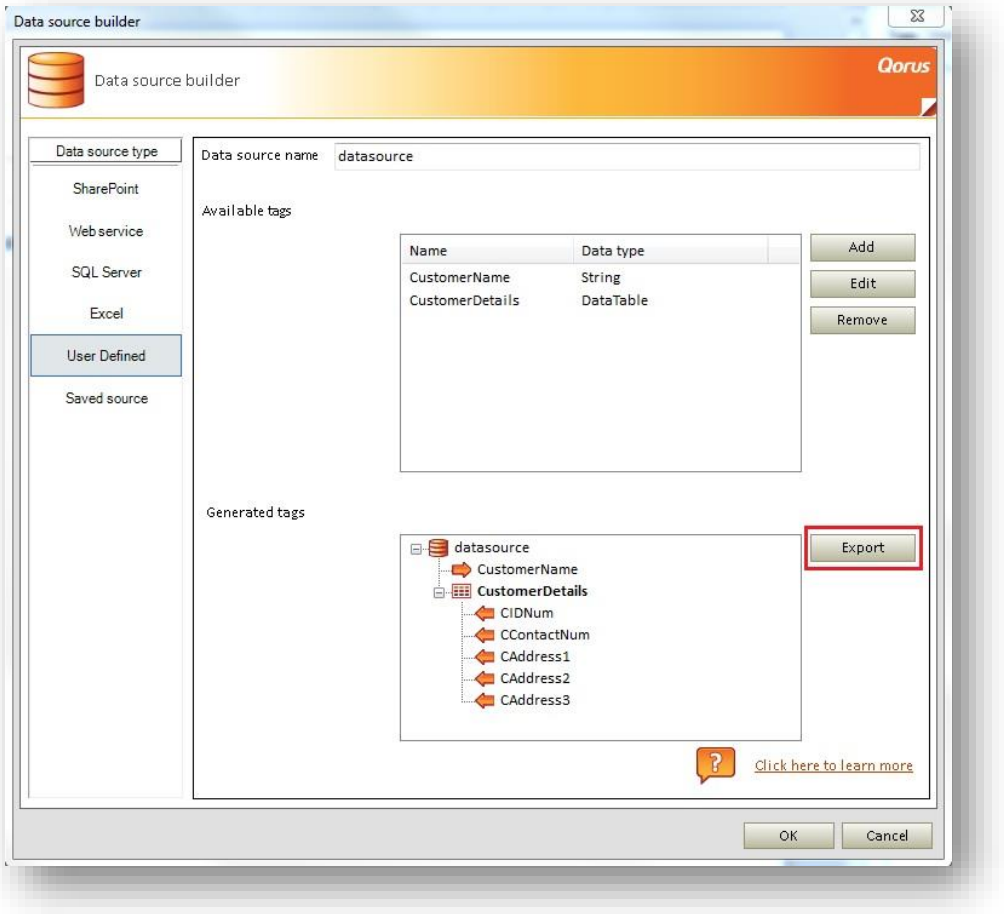
- Edit a Data Tag that has been created in the table.
- Delete a Data Tag from the table.
- Moves selected Data Tag up in the order.
- Moves selected Data Tag down in the order.
- Once the table is done being created it can be viewed from the Generated Tags pane.



From the Available Tags pane, the created Data Tags can be edited and removed by selecting the Data Tag and clicking the appropriate button.



When the Data Source is finished being constructed, “Ok” can be clicked to add it to the template.



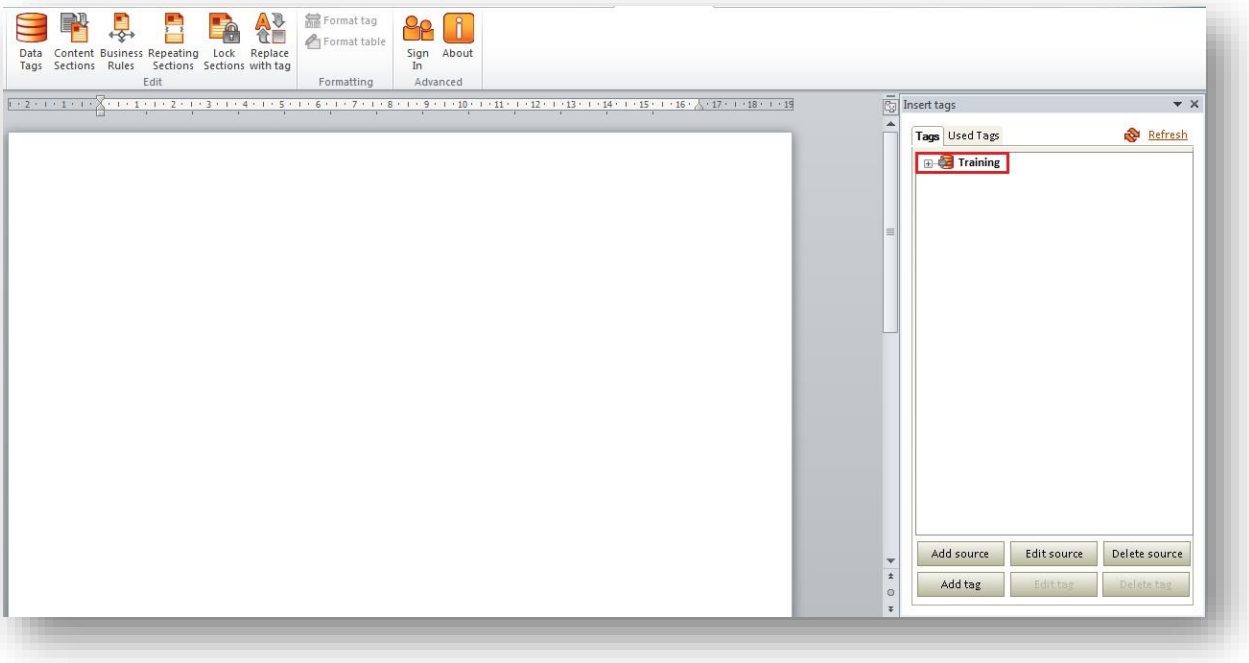
You are now ready to start dragging data tags on the template.

Inserting Data Tags

These tags will be merged with their specific data into the template. Data Tags live within Data Sources.

- Click on the “Data Tags” button in the Qorus ribbon.

- The right side pane will open. From here the user can manage Data Tags and Data Sources.



Note:
The position of the cursor denotes the place where the data tag will be placed when it has been dragged into the template.
For the most accurate placement of the Data Tag, position the cursor to the exact position where the Data Tag is required to be placed and then drag the Data Tag to the cursors position.

- Now, Drag a Data Tag into the template using the mouse. You will see it appear on the template as shown below:

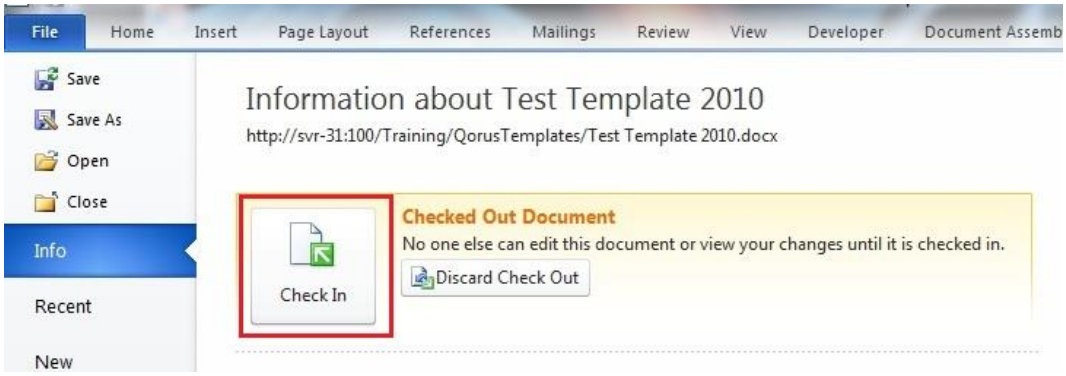
Enter the text you wish you template to contain as well as the tags you require.

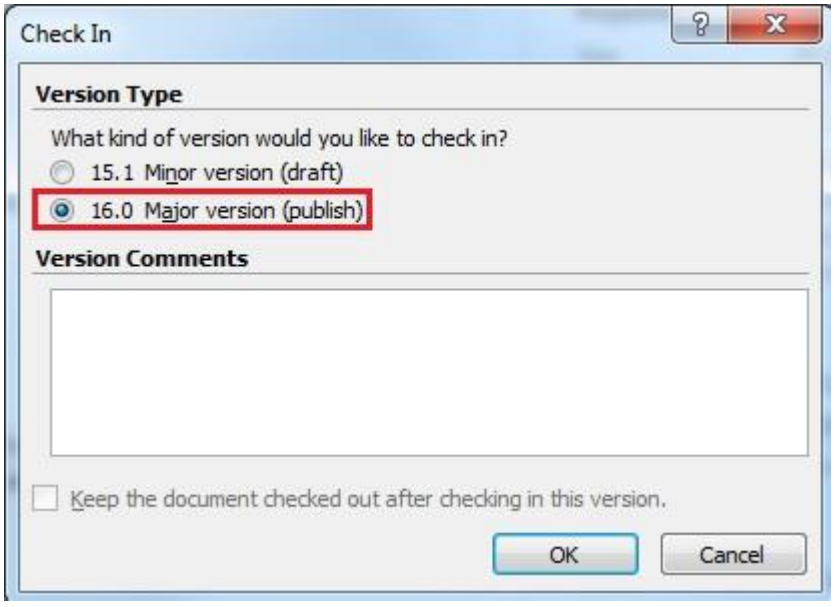
Publish the template

In order to make a template available for merging it has to be saved and published into the library.

When finished editing the template save it like a normal Word document, then it can either be checked in via the open template or via the SharePoint interface.

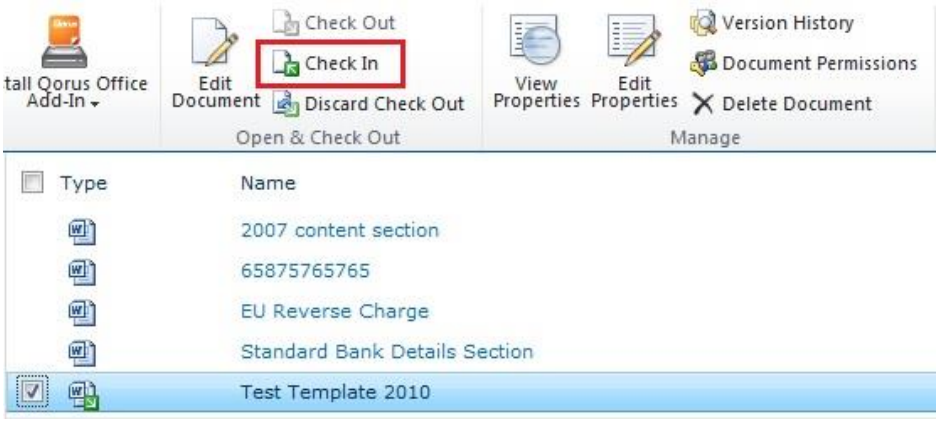
- The template can be checked in via the “File” tab in Word.

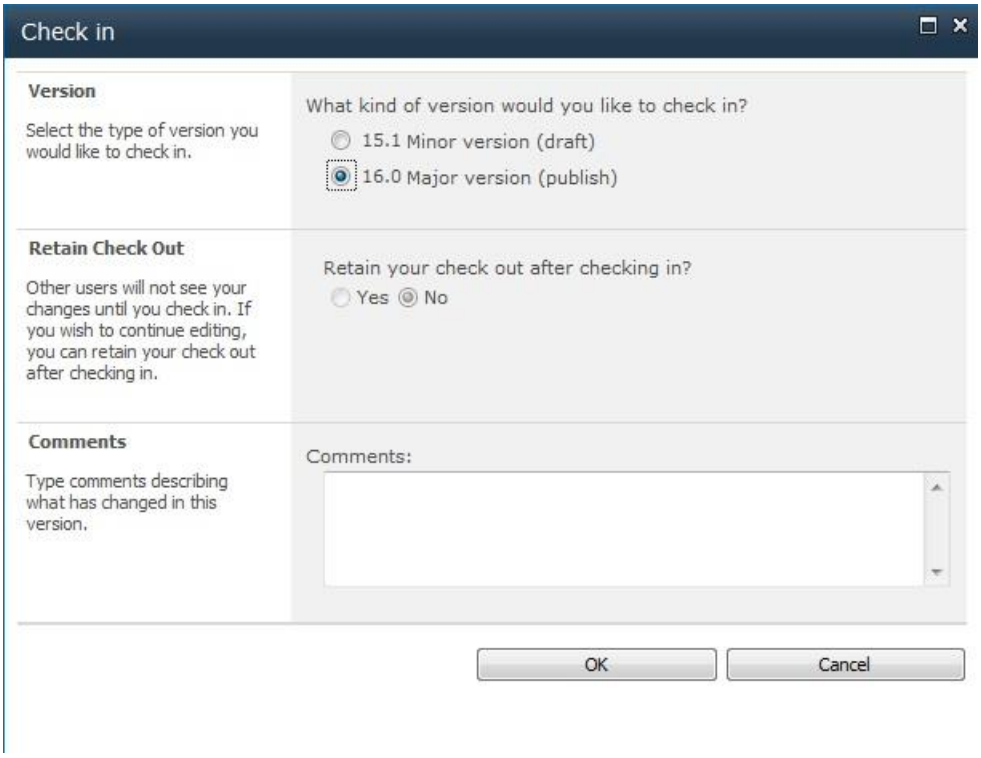




The template can also be published once in SharePoint, via the ribbon.

- Select the template and click the “Check In” button.





Call the merge from the SharePoint API

Your template is now published and is ready to be merged with external data using the SharePoint API.

Refer to ***Qorus DocGeneration SharePoint API*** to connect to the SharePoint API and refer to ***Merging a Qorus template*** to merge the document via the SharePoint API.