How To Create a Twitter Data Mining App With Text To Speech Functionality.

© 2013 *By GertJan de Leeuw | De Leeuw ICT | e-mail: gertjan@deleeuwict.nl*

This tutorial explains the following actions:

1. How to create a TriniDAT App from scratch with Visual Basic.NET.
2. How to map HTTP URLs to pure .NET code.
3. How to send object messages to other mapping point classes.
4. How to manufacture & install TriniDAT apps.

This demo demonstrates how to create a TriniDAT application that does the following:

1.) Display a live tweet based on a keyword search
2.) Speak a tweet with the .NET text-to-speech engine.

Note: This example code was constructed using Microsoft Visual Basic 2010 Express.

Basic Requirements:

TriniDAT Data Application Server
Microsoft Visual Studio 10 |
http://www.microsoft.com/visualstudio/eng#products/visual-studio-2010-express

1. Preparing your VB.NET project.

1.1 Creating a TriniDAT Mapping Point Project in Visual Basic.NET

1. Start your VB.NET environment
2. Click File –> New Project → Class Library.
3. Create a new project named 'JTwitterDemo'.
4. Click OK to create the project.
5. Now we need to configure the project to compile as a TriniDAT webservice.
6. In VB.NET , click on menu Project → JTwitterDemo Properties...
7. Click on the 'References' tab.
8. Click on the 'Add..' button.
9. We'll need to add 2 references to make this work. One reference for coding a TriniDAT mapping point, anotherr for JSON to XML conversion. Browse to the TriniDAT Data Application Server installation folder. Now, select the files TriniDATServerTypes.dll and Newtonsoft.Json.dll.

1.2 Display TriniDAT Mapping Point Sample Code

1. Start TriniDAT Data Application Server.
2. In Simon's console window, enter 'codestub'.
3. Your text-editor should display the webservice template code file (aka 'mapping point\)

4. Copy/paste the source in the .NET project's Class1 code window.
5. Verify that .NET does not report compiler errors.
6. Rename the example's class 'JHelloWorld' to 'JTwitterParser'.It is highly recommend you rename the class by renaming the file in VB's Solution Explorer. It will automatically update the source-code.
7. For this example to work, you'll need to add the following declarations:

    Imports System.Web
    Imports System.Text
    Imports Newtonsoft.Json

    The final source-code stub should look like this:

File: TwitterParser.vb

```
Option Compare Text
Option Explicit On

'TriniDAT Application Server - Webservice Sample Code for COPY/PASTE purposes.
'(c) 2013 GertJan de Leeuw | De Leeuw ICT | www.deleeuwict.nl | Visit the Developer Community Forum for more code examples.

Imports System.Web
Imports System.Text
Imports System.Collections.Specialized
Imports TriniDATServerTypes
Imports Newtonsoft.Json

Public Class JTwitterParser
   Inherits JTriniDATWebService

   Private my_mapping_point As MappingPointBootStrapData

   Public Overrides Function DoConfigure() As Boolean

      'Create local inbox to receive mapping point objects.
      Dim my_mailbox As TriniDATObjectBox_EventTable

      my_mailbox = New TriniDATObjectBox_EventTable
      my_mailbox.event_inbox = AddressOf Me.myInbox
      getMailProvider().Configure(my_mailbox, False)

      'Set-up a bare bone HTTP event table.
      Dim my_http_events As TriniDATHTTP_EventTable

      my_http_events = New TriniDATHTTP_EventTable
      my_http_events.event_onget = AddressOf OnGet
      my_http_events.event_onpost = AddressOf OnPost

      Return getIOHandler().Configure(my_http_events) 'True.
   End Function
```

```
    Public Overrides Function OnRegisterWebserviceFunctions(ByVal servers_function_table As
TriniDATServerFunctionTable) As Boolean

        Dim my_get_function As TriniDAT_ServerGETFunction
        Dim myget_parameter_spec As TriniDAT_ServerFunctionParameterSpec

        'set up dynamic sub-mapping point /helloworld.
        my_get_function = New TriniDAT_ServerGETFunction(AddressOf ShowHelloWorld)
        my_get_function.FunctionURL = Me.makeRelative("/helloworld")

        'Require this URI to be called with at least one parameter.
        myget_parameter_spec = New TriniDAT_ServerFunctionParameterSpec()
        myget_parameter_spec.ParameterName = "myrequiredparameter"
        myget_parameter_spec.ParameterType = "String"
        myget_parameter_spec.Required = True 'default.

        'Add Parameter to the new GET function spec.
        my_get_function.Parameters.Add(myget_parameter_spec)

        'Add this Function to the server's internal HTTP traffic routing-table.
        servers_function_table.Add(my_get_function)

        Return True 'Success.
    End Function

    Public Sub ShowHelloWorld(ByVal parameter_list As
TriniDATServerTypes.TriniDATGenericParameterCollection, ByVal AllParameters As
System.Collections.Specialized.StringDictionary, ByVal Headers As
System.Collections.Specialized.StringDictionary)
        '/helloworld/ dedicated uri code handler.

        Me.getIOHandler().addOutput("Hello world @ dedicated URI.")
        Me.getIOHandler().addOutput("<BR>")
        Me.getIOHandler().addOutput("Value of myrequiredparameter = " &
parameter_list.getById("myrequiredparameter").ParameterValue)

    End Sub

    Public Function myInbox(ByRef msg As JSONObject, ByVal from_url As String) As Boolean

        'Catch mapping point startup messages.
        If msg.ObjectTypeName = "JAlpha" And msg.Directive = "MAPPING_POINT_START" Then

            'Store all mapping point config locally.
            Me.my_mapping_point = CType(msg.Attachment, MappingPointBootstrapData)
            Return False
        End If

        'Catch mapping point shutdown messages.
        If msg.ObjectTypeName = "JOmega" And msg.Directive = "MAPPING_POINT_STOP" Then

            Return False
```

```
        End If

        Return False
    End Function

    Public Sub OnGet(ByVal HTTP_URI_Path As String, ByVal HTTP_URI_Parameters As
StringDictionary, ByVal HTTP_URI_Headers As StringDictionary)

        'Your GET code handler goes here.
        Me.getIOHandler().addOutput("Hello world @ GET handler.")

        'Add JTextToSpeech to your mapping point dependency list to make this object exchange
example work:
        'Dim speak_request As JSONObject

        'speak_request = New JSONObject
        'speak_request.ObjectType = "JTextToSpeech"
        'speak_request.Directive = "SPEAK"
        'speak_request.Attachment = "Somebody just visited my website, isn't that just wonderful."
        'Me.getMailProvider().send(speak_request, Nothing, "JTextToSpeech")

    End Sub

    Public Sub OnPost(ByVal HTTP_URI_Path As String, ByVal HTTP_URI_Parameters As
StringDictionary, ByVal HTTP_URI_Headers As StringDictionary)

        'Your POST code handler goes there.
        Me.getIOHandler().addOutput("Hello world @ POST handler.")
    End Sub

End Class
```

We will now edit this sample code until we have a fully fledged Twitter Parser.

About The Default Mapping Point Code Stub

This might be a good time to become familair with TriniDAT's code example. The code stub can easily be used for any new mapping point project you develop.
The example code declares a sub-url in the mapping point as /myhelloworld with 1 required String parameter called '*myrequiredparameter*'.

If you were to compile & install this example, you could execute it by navigating to http://<serverip>/<appid>/<mapping_point_url>/ helloworld?myrequiredparameter=test or by navigating to http://<serverip>/<appid>/<mapping_point_url>/ . In the OnGet handler, the output in the webbrowser would simply be:

Hello World @ GET handler.

2. The Twitter Demo – Setting Up Callable URLs

The mapping point sourcecode will be designed to serve 2 at URLs:

http://<serverip>/<appid>/<mapping_point_url>/search?keyword=abc

and:

http://<serverip>/<appid>/<mapping_point_url>/speak?keyword=abc

The /speak url will call on the same code as the /search URL. Therefore we must first create the tweet search code.

3. Adding Twitter Search functionality to the JTwitterParser class.

Add this function to the JTwitterParser class. This code connects to the Twitter API server to retrieve a tweet based on the passed keywords.

Since we want to use this code on other places it does not yet return the tweet's message text to the user's web browser.

```
 Public Function twitterSearch(ByVal query As String) As String

        Dim search_url As String
        Dim wc As New WebClient
        Dim jsondata As String
        Dim jxml As XDocument
        Dim user_alias As String
        Dim twitter_result As XElement

        Try
            'retrieve tweet.
            search_url = "http://search.twitter.com/search.json?q=" & query &
"g&rpp=1&include_entities=true&result_type=recent"

            'set encoding.
            wc.Encoding = System.Text.Encoding.UTF8

            'Wrap output in parential nodes to get the correct XML.
            jsondata = "{'?xml': { '@version' : '1.0', '@standalone' : 'no' },'root' : " &
wc.DownloadString(search_url) & " }"

            'Convert JSON -> XML.
            jxml = XDocument.Parse(JsonConvert.DeserializeXmlNode(jsondata).OuterXml)

            'Extract result.
            twitter_result = jxml.<root>.<results>(0)

            If Not IsNothing(twitter_result) Then
                Dim twitter_msg_text As String

                'Parse & clean up tweet
                twitter_msg_text = twitter_result.<text>.Value
                twitter_msg_text = Replace(twitter_msg_text, "...", " ")
                twitter_msg_text = Replace(twitter_msg_text, ";", "")
                twitter_msg_text = Replace(twitter_msg_text, "@", "")
                twitter_msg_text = Replace(twitter_msg_text, "..", " ")
                twitter_msg_text = Trim(twitter_msg_text)
                twitter_msg_text = WebUtility.HtmlDecode(twitter_msg_text)

                user_alias = twitter_result.<from_user_name>.Value 'fullname

                'return formatted tweet text to caller.
                Return user_alias & " says " & Chr(34) & twitter_msg_text & Chr(34)
            Else
                Return "nobody speaks that kind of language"
            End If
```

```
    Catch ex As Exception
        Return "There was an error: " & ex.Message
    End Try
End Function
```

4.Implementing the /search and /speak URLs.

Now that we have created a Twitter search routine, we want to call it from the URLS /search and /speak in this demo app.

Let's implement these URL handlers now. Add these functions to the JTwitterParser class:

/search uri-handler code:

```
Public Sub MyTwitterSearchUrlHandler(ByVal parameter_list As
TriniDATServerTypes.TriniDATGenericParameterCollection, ByVal AllParameters As
System.Collections.Specialized.StringDictionary, ByVal Headers As
System.Collections.Specialized.StringDictionary)

    'set encoding if you want to parse non-english.
    Me.getIOHandler().setEncoding(New UTF8Encoding)

    'write tweet to browser.
Me.getIOHandler().addOutput(twitterSearch(parameter_list.getById("keywords").Parameter
rValue))

End Sub
```

/speak uri-handler code:

```
Public Sub MyTwitterSpeakUrlHandler(ByVal parameter_list As
TriniDATServerTypes.TriniDATGenericParameterCollection, ByVal AllParameters As
System.Collections.Specialized.StringDictionary, ByVal Headers As
System.Collections.Specialized.StringDictionary)

    Dim twitter_text As String
    Dim speak_msg As JSONObject

    twitter_text = twitterSearch(parameter_list.getById("keywords").ParameterValue)

    If InStr(twitter_text, " says ") > 0 Then
       'let's send a message to our friendly neighbor class JTextToSpeech
       '=========================================
       'JTextToSpeech knows everything about TTS stuff and we don't.
       'This perfectly illustrates the power of TriniDAT app development.
       'Our class is specialized in Twitter, while the neighbor  is specialized in tts etc.

       'craft a speak request message.
       '======================

       speak_msg = New JSONObject
       speak_msg.ObjectType = "JTextToSpeech"
       speak_msg.Directive = "SPEAK"

       'Tell it to speak the tweet text.
       speak_msg.Attachment = twitter_text

       Me.getMailProvider().send(speak_msg, Nothing, "JTextToSpeech")

       'Also send the tweet to the browser.
       Me.getIOHandler().addOutput(twitter_text)
    Else
       Me.getIOHandler().addOutput("Error")
    End If

End Sub
```

5. Registering your /search and /speak mapping points with the server.

We need to find tweets based on keyword search. Let's code the routine that does all that.

Note: this demo App is based on the Twitter 1.1 API . As of this date (May 2013 their search api does not require authentication so this code should work from anywhere.

1. Replace the '*OnRegisterWebserviceFunctions*' function in the JTwitterParser class with this code:

```
 Public Overrides Function OnRegisterWebserviceFunctions(ByVal
servers_function_table As TriniDATServerFunctionTable) As Boolean

        Dim twitter_searchurl As TriniDAT_ServerGETFunction
        Dim twitter_search_parameter As TriniDAT_ServerFunctionParameterSpec

        Dim twitter_speak_messageurl As TriniDAT_ServerGETFunction
        Dim twitter_speak_message_parameter As TriniDAT_ServerFunctionParameterSpec

        'add '/search?keywords=..' sub-mapping point.
        twitter_searchurl = New TriniDAT_ServerGETFunction(AddressOf
MyTwitterSearchUrlHandler)
        twitter_searchurl.FunctionURL = Me.makeRelative("/search")

        'Require this URI to be called with at least one parameter.
        twitter_search_parameter = New TriniDAT_ServerFunctionParameterSpec()
        twitter_search_parameter.ParameterName = "keywords"
        twitter_search_parameter.ParameterType = "String"
        twitter_search_parameter.Required = True 'default.

        'Add keyword parameter.
        twitter_searchurl.Parameters.Add(twitter_search_parameter)


        'add '/speak?keywords=..' sub-mapping point.
        twitter_speak_messageurl = New TriniDAT_ServerGETFunction(AddressOf
MyTwitterSpeakUrlHandler)
        twitter_speak_messageurl.FunctionURL = Me.makeRelative("/speak")

        twitter_speak_message_parameter = New TriniDAT_ServerFunctionParameterSpec()
        twitter_speak_message_parameter.ParameterName = "keywords"
        twitter_speak_message_parameter.ParameterType = "String"
        twitter_speak_message_parameter.Required = True

        'Add keyword parameter.
        twitter_speak_messageurl.Parameters.Add(twitter_speak_message_parameter)

        'Register our URLS .
        servers_function_table.Add(twitter_searchurl)
        servers_function_table.Add(twitter_speak_messageurl)

        'Completed.
        Return True
    End Function
```

The *OnRegisterWebserviceFunctions* is called when the server wants to know what URLs you want automatically routed to class functions. Note that this is an optional feature in TriniDAT Data Application Server and that you can also catch all GET and POST requests simply by providing the event_get and event_post with your function addresses in the DoConfigure event. The drawback of this is that you need to manually parse all incoming GET and POST requests.

To let TriniDAT server do the parsing work for you, declare all your functions in the *TriniDATServerTypes.TriniDATSpecializedHTTPHandler* delegate fashion (see code below) and

then declare these functions in TriniDAT server function objects . (variable type s*TriniDAT_Server*Function*).

When your mapping point's URL is invoked by the browser the server will lookup the correct function in its internal routing table. Optionally, you can specify a required parameter list and data types to shield your code from invalid requests.

When using URI routing features, be sure to not forget:

1) Call *servers_function_table.Add()* to add the final declarations to the server's routing table.

2) Ensure the required implementation of the function *OnRegisterWebserviceFunctions* always return true, even when you don't use it. Just like the DoConfigure event, a mapping point will not be execute when this function returns false.

File: TwitterParser.vb Final Version.

```
Option Compare Text
Option Explicit On

Imports System.Collections.Specialized
Imports TriniDATServerTypes
Imports System.Net
Imports System.Text
Imports System.Web
Imports Newtonsoft.Json


Public Class JTwitterParser
    Inherits JTriniDATWebService

    Private my_mapping_point As MappingPointBootStrapData

    Public Overrides Function DoConfigure() As Boolean

        'Create local inbox to receive mapping point objects.
        Dim my_mailbox As TriniDATObjectBox_EventTable

        my_mailbox = New TriniDATObjectBox_EventTable
        my_mailbox.event_inbox = AddressOf Me.myInbox
        getMailProvider().Configure(my_mailbox, False)

        'Set-up a bare bone HTTP event table.
        Dim my_http_events As TriniDATHTTP_EventTable

        my_http_events = New TriniDATHTTP_EventTable
        my_http_events.event_onget = AddressOf OnGet
        my_http_events.event_onpost = AddressOf OnPost

        Return getIOHandler().Configure(my_http_events) 'True.
    End Function

    Public Overrides Function OnRegisterWebserviceFunctions(ByVal
servers_function_table As TriniDATServerFunctionTable) As Boolean

        Dim twitter_searchurl As TriniDAT_ServerGETFunction
        Dim twitter_search_parameter As TriniDAT_ServerFunctionParameterSpec

        Dim twitter_speak_messageurl As TriniDAT_ServerGETFunction
        Dim twitter_speak_message_parameter As TriniDAT_ServerFunctionParameterSpec

        'add '/search?keywords=..' sub-mapping point.
        twitter_searchurl = New TriniDAT_ServerGETFunction(AddressOf
MyTwitterSearchUrlHandler)
        twitter_searchurl.FunctionURL = Me.makeRelative("/search")
```

```
        'Require this URI to be called with at least one parameter.
        twitter_search_parameter = New TriniDAT_ServerFunctionParameterSpec()
        twitter_search_parameter.ParameterName = "keywords"
        twitter_search_parameter.ParameterType = "String"
        twitter_search_parameter.Required = True 'default.

        'Add keyword parameter.
        twitter_searchurl.Parameters.Add(twitter_search_parameter)


        'add '/speak?keywords=..' sub-mapping point.
        twitter_speak_messageurl = New TriniDAT_ServerGETFunction(AddressOf
MyTwitterSpeakUrlHandler)
        twitter_speak_messageurl.FunctionURL = Me.makeRelative("/speak")

        twitter_speak_message_parameter = New TriniDAT_ServerFunctionParameterSpec()
        twitter_speak_message_parameter.ParameterName = "keywords"
        twitter_speak_message_parameter.ParameterType = "String"
        twitter_speak_message_parameter.Required = True 'default.

        'Add keyword parameter.
        twitter_speak_messageurl.Parameters.Add(twitter_speak_message_parameter)

        'Register our URLS .
        servers_function_table.Add(twitter_searchurl)
        servers_function_table.Add(twitter_speak_messageurl)

        'Completed.
        Return True
    End Function

    Public Sub MyTwitterSearchUrlHandler(ByVal parameter_list As
TriniDATServerTypes.TriniDATGenericParameterCollection, ByVal AllParameters As
System.Collections.Specialized.StringDictionary, ByVal Headers As
System.Collections.Specialized.StringDictionary)

        'set encoding if you plan to use non-english.
        Me.getIOHandler().setEncoding(New UTF8Encoding)

        'write tweet to browser.

Me.getIOHandler().addOutput(twitterSearch(parameter_list.getById("keywords").Parameter
rValue))

    End Sub
    Public Sub MyTwitterSpeakUrlHandler(ByVal parameter_list As
TriniDATServerTypes.TriniDATGenericParameterCollection, ByVal AllParameters As
System.Collections.Specialized.StringDictionary, ByVal Headers As
System.Collections.Specialized.StringDictionary)

        Dim twitter_text As String
        Dim speak_msg As JSONObject

        twitter_text = twitterSearch(parameter_list.getById("keywords").ParameterValue)

        If InStr(twitter_text, " says ") > 0 Then
            'let's send a message to our friendly neighbor class JTextToSpeech
            '=========================================
            'JTextToSpeech knows everything about TTS stuff and we don't.
            'This perfectly illustrates the power of TriniDAT app development.
            'Our class is specialized in Twitter, while the neighbor  is specialized in tts etc.

            'craft a speak request message.
            '======================

            speak_msg = New JSONObject
            speak_msg.ObjectType = "JTextToSpeech"
            speak_msg.Directive = "SPEAK"

            'Tell it to speak the tweet text.
            speak_msg.Attachment = twitter_text

            Me.getMailProvider().send(speak_msg, Nothing, "JTextToSpeech")
```

```
            'Also send the tweet to the browser.
            Me.getIOHandler().addOutput(twitter_text)
        Else
            Me.getIOHandler().addOutput("Error")
        End If

    End Sub

    Public Function myInbox(ByRef msg As JSONObject, ByVal from_url As String) As
Boolean

        'Catch mapping point startup messages.
        If msg.ObjectTypeName = "JAlpha" And msg.Directive = "MAPPING_POINT_START"
Then

            'Store all mapping point config locally.
            Me.my_mapping_point = CType(msg.Attachment, MappingPointBootstrapData)
            Return False
        End If

        'Catch mapping point shutdown messages.
        If msg.ObjectTypeName = "JOmega" And msg.Directive = "MAPPING_POINT_STOP"
Then

            Return False
        End If

        Return False
    End Function

    Public Sub OnGet(ByVal HTTP_URI_Path As String, ByVal HTTP_URI_Parameters As
StringDictionary, ByVal HTTP_URI_Headers As StringDictionary)

        'Your GET code handler goes here.
        Me.getIOHandler().addOutput("Hello world @ GET handler.")

        'Add jTextToSpeech to your mapping point dependency list to make this object
exchange example work:
        'Dim speak_request As JSONObject

        'speak_request = New JSONObject
        'speak_request.ObjectType = "jTextToSpeech"
        'speak_request.Directive = "SPEAK"
        'speak_request.Attachment = "Somebody just visited my website, isn't that just
wonderful."
        'Me.getMailProvider().send(speak_request, Nothing, "jTextToSpeech")

    End Sub

    Public Sub OnPost(ByVal HTTP_URI_Path As String, ByVal HTTP_URI_Parameters As
StringDictionary, ByVal HTTP_URI_Headers As StringDictionary)

        'Your POST code handler goes there.

        Me.getIOHandler().addOutput("Hello world @ POST handler.")

    End Sub

    Public Function twitterSearch(ByVal query As String) As String

        Dim search_url As String
        Dim wc As New WebClient
        Dim jsondata As String
        Dim jxml As XDocument
        Dim user_alias As String
        Dim twitter_result As XElement

        Try
            'retrieve tweet.
            search_url = "http://search.twitter.com/search.json?q=" & query &
"g&rpp=1&include_entities=true&result_type=recent"
```

```
            'set encoding.
            wc.Encoding = System.Text.Encoding.UTF8

            'Wrap output in parential nodes to get the correct XML.
            jsondata = "{'?xml': { '@version' : '1.0', '@standalone' : 'no' },'root' : " &
wc.DownloadString(search_url) & " }"

            'Convert JSON -> XML.
            jxml = XDocument.Parse(JsonConvert.DeserializeXmlNode(jsondata).OuterXml)

            'Extract result.
            twitter_result = jxml.<root>.<results>(0)

            If Not IsNothing(twitter_result) Then
               Dim twitter_msg_text As String

               'Parse & clean up tweet
               twitter_msg_text = twitter_result.<text>.Value
               twitter_msg_text = Replace(twitter_msg_text, "...", " ")
               twitter_msg_text = Replace(twitter_msg_text, ";", "")
               twitter_msg_text = Replace(twitter_msg_text, "@", "")
               twitter_msg_text = Replace(twitter_msg_text, "..", " ")
               twitter_msg_text = Trim(twitter_msg_text)
               twitter_msg_text = WebUtility.HtmlDecode(twitter_msg_text)

               user_alias = twitter_result.<from_user_name>.Value 'fullname

               'return formatted tweet text to caller.
               Return user_alias & " says " & Chr(34) & twitter_msg_text & Chr(34)
            Else
               Return "nobody speaks that kind of language"
            End If

        Catch ex As Exception
            Return "There was an error: " & ex.Message
        End Try
    End Function

End Class
```

6. Debugging your TriniDAT applications.

In the present version, TriniDAT Data Application debugging is limited to the application manifest level only. If you want to edit the application index in real-time you'll need to ensure the server's executable has sufficient file permissions on the file apps\index.xml. Realtime DLL rebuilding and reloading will not be possible due to optimalization induced file locks. At any time, a developer may however reload the application manifest by entering 'RELOAD' at the console windows.

The RELOAD command will flush the application cache and will reloads all manifest xml files. This allows a developer or administrator to extend mapping points with new functionality (e.g classes) without the need to restart TriniDAT Data Application server.

In short, it means that a .NET developer can change the application's xml manifest file, but not rebuild a DLL and load it without restarting the TriniDAT Data Application Server.

7. Creating The Application Manifest File.

Now that your app is done you want to test it out. At this point, the code development stage is completely done, and all that's left to do is to make an application manifest file. This file basically contains information about your .NET class library and application meta-data info.

The application manifest is created in 8 simple steps.

2. Make sure you compiled the JTwitterDemo project and have output DLLs.
3. In your .NET project folder, create a new sub folder called 'TriniDAT'.
4. Copy your compiled DLL to this folder.
5. Start TriniDAT Data Application Server.
6. Type 'appstub' in Simon's console window.
7. Save the file on your harddrive as a new file called AppManifest.xml in the 'TriniDAT' folder.
8. Now edit the application manifest:
    1. Set the 'App' node's 'name' attribute to 'Twitter Parser Demo'
    2. Find the APP/MP node and change the 'url' attribute to "/twitter". This sets the mapping point URL from which your application will be available from a client's web browser
    3. Go to the APP/MPS/MP section.
    4. Change the template 'Jclass' node's 'id' to 'JTwitterDemo'. This tells TriniDAT server about your .NET classname.
    5. Add a new Jclass node with id 'JTextToSpeech'. This will make this class available for calling in our mapping point project.
    6. The classname has been set. Now we need to report DLL filepaths so the TriniDAT class loader knows where to load your .NET classes from.
    7. Under APP/MPS/MP/DEPENDENCIES node, set the 'dependency' node's path attribute to the path of your class library DLL file*. (example: "C:\TriniDAT\JTwitterDemo\bin\Release\ JtwitterDemo.DLL .Any dependency information you enter in this section will be used to resolve missing types.
    8. Save the manifest file. You can close your text-editor.

*** Note about how TriniDAT Data Application Server Loads .NET DLLs:**

1. It's not necessary to declare dependency info for classes prefixed with TriniDAT, native TriniDAT classes ('JTextToSpeech', 'JInteractiveConsole' and 'JWebBrowser') native .NET classes or DLLs registered in .NET's GAC (*Global Application Cache*).
2. If you plan to distribute your apps to another computer (such as selling it in the appstore) then you shou;d not make use of any GAC string. For fluent TriniDAT app distrubution you are required to declare all class information interms of filepaths.TriniDAT Data App server does not depend on the GAC to resolve type information because this information is different on each computer.A commercial App developer should make sure all of his .NET types can be resolved by the filepath information provided in the application's manifest xml file.
3. If in any case you are required to load a type from the GAC you can pass a full GAC string in the 'path' attribute of a dependency node.(for example path="*Microsoft.mshtml, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a*")

AppManifest.xml :

```
<app name="JTwitterDemo Application" interface="false" author="My Name"
authorcontactwebsite="http://www.deleeuwict.nl" runtimecount="0">
  <mps>
    <mp url="/twitter" description="My Webservice" enablestats="true"
interactiveconsolefeatures="true">
      <jclass id="JTextToSpeech"/>
```

```
        <jclass id="JTwitterParser" />

    <dependencies>
      <dependency path="JTwitterDemo.DLL" />
    </dependencies>
   </mp>
  </mps>
</app>
```

Important: Notice the fine distinction between a .NET project library name and the webservice class name. The release version DLL will typically have the same name as the class library project but in the appmanifest you must specify the webservice class name in the <dependency> node.

How to install an app in TriniDAT Data Application Server's application cache.

You can get your app up and running in two different ways. The prerequirement either way case is that you need to create an application manifest file that will describes your application in XML to the application server's mapping point loader

1) Manually changing the server's application index file (file *\trinidat_config\apps\apps.xml*) to point to a different manifest file.

2) ZIP all your application files and then installing this archive by clicking on 'Install App'.

The first method will be useful if you desire to set the application's appid or to hard-code a manifest file's location yourself. Otherwise, these functions are automatically managed by the server. Be sure to issue an RELOAD command after manually editting server files.

A creator of TriniDAT apps should edit the application's manifest file in such a way that all his classes are loaded from the .NET development environment. These are typically the Visual Studio .NET bin\release and bin\debug folders.

Note: if you publish your applications to the TriniDAT application store then whatever DLL are declared in the manifest file will be copied to our server, so make sure that your final application manifest points to the right DLLs.

9. Packaging the application for installation.

After developing your app iyou can install it in TriniDAT Data Application Server as an application running on its own application URL. Applications that you have manually added by editing the server's application index may also be packaged automatically by issuing the PUBLISH <appid> command. Note that this command will upload your application straight to the appstore. Follow these steps to package your app on your local machine.

1.Create a new ZIP file named MyApp.zip. Add the following files:

AppManifest.xml

JtwitterDemo.dll

2. Go to TriniDAT Data Application Server
3. Click on 'Install App..' button.
4. Select the newly created app package.
5. The server will now install the packaged app. A unique ID will be assigned to your application and registered in the application server index.
6. When the installation is completed, Simon will output the new application ID. This ID is at all times the server's own reference to your application and mapping point URLs.

11. Obtaining the full server URL to your TriniDAT Application.

7. The fast way: Click on the 'WWW' button to navigate to the application index page. Your app should appear here. You can also issue the '*WWW*' command for the same effect.
8. You can also type 'appinfo <appid>' in Simon's console to display all information.
9. The APPINFO command reports both absolute and relative URLS.

12. Testing Your Twitter Enabled App.

Navigate to your application's url in your browser, for example:

Testing the /SEARCH mapping point URL:

http://192.168.2.1/7/twitter/search?keywords=food

The app outputs a random tweet containing the word '*food*'.

Testing the /SPEAK mapping point URL:

http://192.168.2.1/7/twitter/speak?keywords=meat

The app both speaks and outputs a random tweet containing the word '*meat*'.