**Workflow Client API (SourceCode.Workflow.Client.dll)**

* The workflow client API refers to the .NET assembly **SourceCode.Workflow.Client.dll.**
* **This** assembly exposes several methods and classes that we can use when interacting programmatically with deployed workflow.
* you can find the assembly at the following location:

**[Program Files (x86)]\K2 blackpearl\Bin\SourceCode.Workflow.Client.dll**

* In practice, the SourceCode.Workflow.Client assembly is often used to
  + create custom task lists,
  + to start workflows or
  + complete workflow tasks (known as worklist items) from custom user interfaces like ASP.NET web pages,
  + and to create alternative interfaces or tools to allow users to maintain their out-of-office status and worklist items.
  + The workflow client is also often used to create automated testing code or applications that are executed against a workflow definition to verify that the workflow is behaving as expected.

In addition to the Sourcecode.Workflow.Client assembly it is usually necessary to reference the assembly SourceCode.HostClientAPI.dll as well, since this assembly is used to construct more advanced K2 connection strings.

The Workflow.Client assembly is installed in the GAC of the K2 server and on developer workstations when the K2 Studio/K2 for Visual Studio or K2 Core components are installed.

The class library reference is grouped into the following sections:

1. **Design-time and Authoring assemblies**: assemblies used to create K2 artifacts like workflows and SmartObjects.
2. **Runtime Assemblies**: assemblies used to interact with deployed artifacts or the K2 environment at runtime.
3. **Other Assemblies**: assemblies that do not fall into Design-time or Runtime categories.

|  |  |
| --- | --- |
|  |  |

Workflow Client Connections

* Establishing a connection with the Workflow.Client class is the first step to perform workflow runtime interaction.
* The Connection object is the entry point for subsequent calls to the K2 server when using the Workflow Client assemblies, and once you have opened a connection, you can perform actions like retrieving worklists, completing worklist items or starting process instances.
* Under the covers, the Connection object uses RPC calls to the K2 server to execute methods with a proprietary message format. By default, communication to the K2 server when using the Workflow Client API is via Port 5252, although you may have selected a different port when installing K2

1. The simplest way to open a connection is just to specify the K2 server name, in which case the connection to K2 will be established as the current Windows user account (so-called Integrated authentication).

* Simple Connection:

using the current user’s Active Directory credentials and opening a connection on the default 5252 port.

//simple usage of SourceCode.Workflow.Client.Connection  
SourceCode.Workflow.Client.Connection WorkflowClientConnection = new SourceCode.Workflow.Client.Connection();  
WorkflowClientConnection.Open("[ServerName]");  
  
//do things with the connection  
  
//connection must be closed once you are done  
WorkflowClientConnection.Close();

🡪Advanced connection:

If you need to build up a more advanced connection (for example a different security provider, username or port)

import the **SourceCode.Hosting.Client** namespace from the **SourceCode.HostClientAPI.dll** assembly and instantiate a **ConnectionStringBuilder** object to construct a more advanced connection string.

This is especially important when you need to pass through alternative security credentials or are using a custom security provider to authenticate a non-Active Directory user account.

//Make a connection that authenticates against the K2 SQL User Manager  
SourceCode.Hosting.Client.BaseAPI.SCConnectionStringBuilder builder = new SCConnectionStringBuilder();  
builder.Authenticate = true;  
builder.Host = "localhost"; //server name of the K2 host server  
builder.Port = 5555; //use port 5252 for SourceCode.Workflow.Client connections  
builder.Integrated = false;  
builder.IsPrimaryLogin = true;  
builder.SecurityLabelName = "K2SQL"; //the name of the security label to use for authenticating the credentials below  
builder.UserID = "username"; //user name to be authenticated  
builder.Password = "password"; //password for the user to be authenticated  
//open the connection  
SourceCode.Workflow.Client.Connection WorkflowClientConnection = new SourceCode.Workflow.Client.Connection();  
WorkflowClientConnection.Open("[ServerName]", builder.ToString());  
//do things with the connection  
//connection must be closed once you are done  
WorkflowClientConnection.Close();

Connection String Samples

This topic provides some typical samples of connection strings

These examples assume the default blackpearl security labels are in place, where the 'K2' security label equates to Active Directory and the 'K2SQL' security label works with the SQL User Manager

For claims authentication and custom user managers, the following properties in the connection string remain the same as for windows authentication:  
host=**{MachineName}**;  
port=**{K2PortNumber}**;  
Integrated=**true**;  
IsPrimary=**true**;  
Authenticate=**true**

**Example 1: Make a connection under the currently Windows Identity that is running this code**

//Make a connection under the currently Windows Identity that is running this code  
SourceCode.Hosting.Client.BaseAPI.SCConnectionStringBuilder builder = new SCConnectionStringBuilder();  
builder.Authenticate = true; //whether to authenticate the user's credentials against the security provider, usually true  
builder.Host = "localhost"; //name of the K2 host server, or the name of the DNS entry pointing to the K2 Farm  
builder.Port = 5555; //use port 5252 for SourceCode.Workflow.Client connections, port 5555 for everything else  
builder.Integrated = true; //true = use the logged on user, false = use the specified user  
builder.IsPrimaryLogin = true; //true = re-authenticate user, false = use cached security credentials  
builder.SecurityLabelName = "K2"; //the name of the security label to use for authentication

**Example 2: Force the connection under a specific AD Identity**

//Force the connection under a specific AD Identity  
SourceCode.Hosting.Client.BaseAPI.SCConnectionStringBuilder builder = new SCConnectionStringBuilder();  
builder.Authenticate = true; //whether to authenticate the user's credentials against the security provider, usually true  
builder.Host = "localhost"; //server name of the K2 host server  
builder.Port = 5555; //use port 5252 for SourceCode.Workflow.Client connections  
builder.Integrated = false; //true = use the logged on user, false = use the specified user  
builder.IsPrimaryLogin = true; //true = re-authenticate user, false = use cached security credentials  
builder.SecurityLabelName = "K2"; //the name of the security label to use for authenticating the credentials below  
builder.WindowsDomain = "domain"; //when using AD, the name of the domain for the user to be authenticated  
builder.UserID = "username"; //user name to be authenticated  
builder.Password = "password"; //password for the user to be authenticated, unencrypted

**Example 3: Make a connection that authenticates against the K2 SQL User Manager**

//Make a connection that authenticates against the K2 SQL User Manager  
SourceCode.Hosting.Client.BaseAPI.SCConnectionStringBuilder builder = new SCConnectionStringBuilder();  
builder.Authenticate = true; //whether to authenticate the user's credentials against the security provider, usually true  
builder.Host = "localhost"; //server name of the K2 host server  
builder.Port = 5555; //use port 5252 for SourceCode.Workflow.Client connections  
builder.Integrated = false; //true = use the logged on user, false = use the specified user  
builder.IsPrimaryLogin = true; //true = re-authenticate user, false = use cached security credentials  
builder.SecurityLabelName = "K2SQL"; //the name of the security label to use for authenticating the credentials below  
builder.UserID = "username"; //user name to be authenticated  
builder.Password = "password"; //password for the user to be authenticated

**Example 4: Using the connection string**

Once you have constructed the connection string, you can use it to establish a connection to K2. Here are same examples:

//Using the connection string  
  
//SourceCode.Workflow.Client  
SourceCode.Workflow.Client.Connection WorkflowClientConnection = new Connection();  
WorkflowClientConnection.Open("localhost", builder.ConnectionString);  
  
//SourceCode.Workflow.Management  
SourceCode.Workflow.Management.WorkflowManagementServer WorkflowManagementConnection = new WorkflowManagementServer();  
WorkflowManagementConnection.Connection.Open(builder.ConnectionString);  
  
//SourceCode.SmartObjects.Client  
SourceCode.SmartObjects.Client.SmartObjectClientServer SmartObjectClientServerConnection = new SmartObjectClientServer();  
SmartObjectClientServerConnection.CreateConnection();  
SmartObjectClientServerConnection.Connection.Open(builder.ConnectionString);

|  |  |
| --- | --- |
|  |  |

1. Once the connection is open, the process instance can be accessed and used. Once the required interaction with a process has finished the connection must be closed using the **connection.Close()** method. The recommended approach is to use **try{...} catch{...} finally{...}** blocks or **using{...}** blocks to ensure that connections are closed once they are no longer needed. The code samples below illustrate some ways you can manage connections when using the Workflow Client classes.

//managing connections  
  
//manually closing the connection  
SourceCode.Workflow.Client.Connection WorkflowClientConnection = new SourceCode.Workflow.Client.Connection();  
WorkflowClientConnection.Open("[ServerName]");  
//do something with the connection  
//close the connection when it is no longer needed  
WorkflowClientConnection.Close();  
  
  
//using a try..catch..finally block  
SourceCode.Workflow.Client.Connection WorkflowClientConnection1 = new SourceCode.Workflow.Client.Connection();  
WorkflowClientConnection1.Open("[ServerName]");  
try {  
 //do something with the connection  
} catch {  
 //do something with exceptions  
} finally {  
 //make sure the connection is closed when it is no longer needed  
 WorkflowClientConnection1.Close();  
}  
  
  
//using a using block will call close and dispose when the using statement exits  
using (SourceCode.Workflow.Client.Connection WorkflowClientConnection2 = new SourceCode.Workflow.Client.Connection()) {  
 WorkflowClientConnection2.Open("localhost");  
 //do something with connection once opened  
}

1. Starting an instance of a workflow

This example shows how to use the Workflow Client to start a new workflow instance, including examples of setting data field values in the workflow.

 This sample code requires references to the assemblies:

**SourceCode.HostClientAPI**

**SourceCode.Workflow.Client**

using (SourceCode.Workflow.Client.Connection K2Conn = new Connection()) {  
 //simple connection used, for simplicity  
 K2Conn.Open("localhost");  
  
 //create a new process instance object, using the full name of the workflow  
 SourceCode.Workflow.Client.ProcessInstance K2Proc = K2Conn.CreateProcessInstance(@"[project]\[folder]\[workflowname]");  
  
 //Set some properties for the process instance  
 //setting the folio  
 K2Proc.Folio = "ProcessFolio";  
 //setting datafields (datafields are accessed and set by name.)  
 //Take care to specify a value of the correct data type for the targeted field  
 K2Proc.DataFields["StringDataField"].Value = "somevalue";  
 K2Proc.DataFields["IntegerDatafield"].Value = 1;  
 //XML fields are set using a XML-formatted string  
 System.Xml.XmlDocument xmlDoc = new XmlDocument();  
 //do some work with the XML document  
 //pass the XML document to the XML field as a string  
 K2Proc.XmlFields["XMLDataField"].Value = xmlDoc.ToString();  
  
 //you can iterate over the data fields collection as well  
 foreach(DataField data Field in K2Proc.DataFields) {  
  //do something with each datafield, e.g. read name or set value  
  string fieldName = dataField.Name;  
  string fieldValue = dataField.Value.ToString();  
 }  
  
 //Start the process instance  
 K2Conn.StartProcessInstance(K2Proc);  
  
 //Starting a process instance synchronously  
 K2Conn.StartProcessInstance(K2Proc, true);  
  
 //If needed, read the resulting process instance ID  
 int ProcessInstanceId = K2Proc.ID;  
}

DataField Names in a processinstance object are not strongly-typed. You will need to read and write DataFields by passing the name of the field as a string-indexed value. Ensure that the name you pass exactly matches the name of the datafield as defined in the workflow definition.

**Starting an instance of a workflow synchronously**

* The **StartProcessInstance** method has an overload method that accepts a parameter called Sync.
* This setting effectively determines how quickly K2 will respond with a success message after the workflow has started.
* If you start a workflow Synchronously (in other words, the Sync parameter’s value is True), K2 will not respond with a success message until the next wait-state in the workflow is reached. A wait-state could be something like an activity with a client event, or an activity with a start rule.
* Essentially, this parameter tells K2 to perform all the activities and events in the workflow until the workflow is waiting for something or someone.
* Only when this wait-state is reached will the method return to the calling application. When starting a workflow is this manner, you are ensured that all the intermediate workflow activities and events leading up to the wait state have been completed.

This Synchronous style of execution is mostly used in screen-flow applications or automated testing, because you want to be sure that the first client event is created before the StartProcessInstance call returns. When the call returns, you would normally search through the user’s worklist for the resulting task and then perform the next action, rather than polling the user's worklist several times and waiting for the task to appear.

* In the majority of cases you will want to start a workflow asynchronously because you do not need to wait for activities or events to complete. This is also the default behavior, unless you specifically override by setting the Sync parameter to true, as shown in the following code snippet:

//Starting a process instance synchronously  
K2Conn.StartProcessInstance(K2Proc, true);

Starting workflows synchronously (Sync=true) is slower than starting workflows asynchronously (Sync=false), and you should not use the Synchronous-style execution unless there is a specific reason to do so.