Quick Apps for SharePoint 2013 v6.1

System Integrator Developer Guide

QUEST SOFTWARE®

If you have any questions regarding your potential use of this material, contact:

Quest Software World Headquarters
LEGAL Dept
5 Polaris Way
Aliso Viejo, CA 92656 USA
www.quest.com
email: legal@quest.com

Refer to our Web site for regional and international office information.

Quick Apps for SharePoint System Integrator Developer Guide
Updated - March 2013
Software Version - v6.1

CONTENTS

About this Guide

This guide explains on how to write a custom Data Access Object (DAO) Provider and Configurator for System Integration (SI) Web Parts. If you would like more information on how to set up the SI Web Parts, refer to the Computer Based Training (CBT) for the Quick Apps for SharePoint 2013.

About System Integration (SI) Web Parts

The SI Web Parts enable you to create, edit, and display data from various external systems. The SI Web Parts can use various protocols to communicate with different external systems. SI Web Parts handle communication with external systems (such as Microsoft Dynamics, SAP, Salesforce.com) by using a Data Access Object (DAO) Provider. Out of the box, the SI Web Parts support a DAO Provider that connects with Web Services returning the following types of objects:

- Data Set (System.Data.DataSet)
- Data Table (System.Data.DataTable)
- Data View (System.Data.DataView)
- Array of Data Set. The SQL Server HTTP Endpoint returns this kind of data. SQL Server HTTP Endpoint is the method used to expose your stored procedure as Web Services. See the CBT for the Quick Apps for SharePoint 2013 on how to setup HTTP Endpoint for SQL Server.
- Array that can be serialized into XML and then converted into Data Set.
- XML String that can be converted to Data Set.
- XML Node that can be converted to Data Set.

If you want to connect the SI Web Parts with another system that does not support the mentioned Web Services or interface, the SI Web Parts supports an Application Programming Interface (API) you can use to create a custom DAO Provider.

The SI Web Parts include a Web Part configurator that makes configuring the SI Web Parts much easier than if you had to use the Web Parts property panel. However, there are cases when you want to create a custom configurator for your user. For example, say you have a Web Service method that takes an XML string as parameter. If constructing the XML string is a challenge for some of your users, you can create a custom configurator that asks your users the right question and constructs the XML string for them.

Customizing SI Web Parts

SI Web Parts allows you to customize and build additional connectors and connector configurations into the system. You can create your own custom DAO Provider to connect to other external systems, and create your own custom configurator to build the metadada for the custom DAO Provider to connect to external systems. The configurator supports the Catalog property for SI Web Parts for SQLServer, Oracle, Webservices, and K2.

Customizing SI Web Parts involves the following:

- Using a Custom DAO Provider
- Writing a Custom DAO Provider
- Using a Custom Configurator
- Writing a Custom Configurator

Using a Custom DAO Provider

In order to use a custom DAO Provider, you need to specify the fully qualified class name of your custom DAO Provider in the SI Configuration File.

To use the custom DAO Provider

1. Open the Configuration Editor, and select Custom as the System Type.
2. Enter the syntax of the fully qualified class name:
   `NameSpace.ClassName, AssemblyName, Version=versionNumber, Culture=cultureName, PublicKeyToken=publicKeyToken`.
   For example:
   `MyCompany.SI.CustomDAOProvider, MyCompany.SI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=451cac61f7ec4225`.

Writing a Custom DAO Provider

We include a sample Custom DAO Provider when you install the product. You can access the sample code by using the shortcut in Start | All Programs | Quest Software | Quick Apps for SharePoint | Sample Code. You need Microsoft Visual Studio 2005 to open up the Solution file. The sample custom DAO Provider class can be found in CustomDAOProvider.cs file in the SampleDAOProviderAndConfigurator project.

A custom DAO Provider class must inherit from the WA.Core.Data.DAOProvider class. The WA.Core.Data.DAOProvider class is located in the WA.Core assembly. There is one virtual method that you must override — the Execute method. This method is called when the SI Web Parts need to execute an operation in the external system.

Each SI Web Parts has a Catalog property. Here is an example of the Catalog property:

```
<Catalog>
   <Entity Name="Product" System="AdventureWorks" Service="SIDemo">
      <Operation Method="siGetProductByLocation" Name="ProductList">
         <Parameter Name="LocationID" Type="System.Data.SqlTypes.SqlInt32" Usage="In"
Source="HttpRequest" SourceName="LocationID" />
      </Operation>
   </Entity>
</Catalog>
```

The Catalog property defines the Entities in your external system and the Operations that can be applied on each entity. When the SI Web Part needs to load the data, it will call the default operation of the default entity. The user can also invoke an operation using the toolbar buttons or context menu items that are specified in the Actions property of the Web Parts. In any case, when the operation is invoked, the SI Web Parts will call the Execute method of your custom DAO Provider to do the real job.

The Execute method of the custom DAO Provider takes 2 arguments:

1. serviceInfo – this parameter is of SystemInfo type and represents the service that is specified in the Entity element in the Catalog. In the Catalog property example above, the Entity element refers to AdventureWorks System and SIDemo Service. Your SI Configuration File must contain the definition for this System and Service. The ServiceInfo type is the data structure that represents the Service element in the SI Configuration File. You can retrieve information for the service, such as the service name and properties, using the ServiceInfo type.
2. operationInfo – this parameter is of OperationInfo type and represents the operation that is being invoked. The OperationInfo type is the data structure that represents the Operation element in the Catalog property. You can retrieve various information about the operation, such as the name of the operation, the parameters for the operation, and how to construct the parameter.

Using a Custom Configurator

In order to use a custom configurator, you need to specify the fully qualified class name of your custom configurator in the SI Configuration File.

To use a custom configuration

1. Open up your SI Configuration File using the Configuration Editor
2. Find the system node where you want to add the custom configurator.
3. Right-click the **Properties** node right under that system.
4. Click **Add Property**
5. Enter CustomConfigurator for the Property Name.
6. Enter System.String for the Property Type.
7. Enter the fully classified class name of your custom configurator for the Property Value.

   This is the syntax of a fully qualified class name:

   NameSpace.ClassName, AssemblyName, Version=versionNumber, Culture=cultureName, PublicKeyToken=publicKeyToken

   For example:

   MyCompany.SI.CustomConfigurator, MyCompany.SI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=451cac61f7ec4225

After you save the above configuration, you can invoke your custom configurator.

To invoke your custom configurator

1. Click the Web Part menu of one of the SI Web Parts.
2. Select **Configure | SystemName.**

The Configure > Default menu item will invoke the built-in web part configurator. Any custom configurator that is specified in the SI Configuration File will be listed underneath the Default.

You should be able to use the Default configurator to configure the SI Web Parts for any external system. However, you should use the more specialized custom configurator for the target system if one is available.

Writing a Custom Configurator

A sample Custom Configurator is included when you install the product. You can access the sample code by using the shortcut in Start | All Programs | Quest Software | Quick Apps for SharePoint | Sample Code. You need Microsoft Visual Studio 2005 to open the Solution file. The sample custom configurator class can be found in CustomConfigurator.cs, Page1Action.cs and Page2Action.cs files in the SampleDAOProviderAndConfigurator project.

When explaining custom configurations, we will use the term Configurator Framework. Configurator Framework is the mechanism that is used in the SI Web Parts to invoke the custom configurator specified in the SI Configurator File.

You must provide the following items when creating a custom configurator:

1. A class that implements WA.Core.Plugin.ISIConfiguratorPlugin interface. This class will become the entry point for your custom configurator. The Configurator Framework will invoke this class to retrieve the Map file.
2. A Map file, which is an XML file that contains the definition for the actions and views for your custom configurator. A Map file also contains the information about the flow from one action to another when a certain command is invoked within the custom configurator.

3. One or more Action classes derived from the WA.Core.Configurator.DefaultAction class. The Action class is actually the class that renders the user interface for the configurator and constructs the properties for the web part.

WA.Core.Plugin.ISIConfiguratorPlugin Interface

There are several functions and properties that you must implement when using this interface:

1. string ID { get; }

   This property gets a unique ID for this configurator.

2. ConfiguratorMetadata Configuration { get; }

   This property gets the instance of the Map file. The ConfiguratorMetadata is a class that represents the Map file. You can create the ConfigurationMetadata instance by using the Metadata.LoadConfigurationMetadata function.

3. bool IsSupported(System.Web.UI.WebControls.WebParts.WebPart webPart);

   This function determines whether your configurator supports a specific kind of web part. If your custom configurator supports all the SI Web Parts, you can return true from this function. Otherwise, you can check the type of the web part and return true or false accordingly.

4. SystemInfo System { get; set; }

   This property gets and sets the instance of the SystemInfo object. The SystemInfo type represents the System element in the SI Configuration File that contains your custom configurator.

Map File

A Map file is an XML file that contains the definition of actions and views for your custom configurator. The Map file also contains the flow information from one action to another when a command is invoked in the Configurator Framework.

Here is the syntax of a Map file:

```
<?xml version="1.0" encoding="utf-8" ?>
< Configurator>
   <Actions>
      <Action name="actionName" type="Fully Qualified Class Name of the action class">
         <View name="View Name">
            <Forward name="commandName" action="actionName" view="viewName" />
            <Forward  />
            <Forward  />
         </View>
         <View></View>
         <View></View>
      </Action>
      <Action>.</Actions>
      <Action>.</Actions>
   </Actions>
<Configurator>
```

The root element of the map file is Configurator element. The Configurator element can contain one Actions element. The Actions element can contain one or more Action element. The Actions element can contain one or more View elements. The View element can contain zero or more Forward element.

The following are the attributes for the Action element:

1. name – this is the name of the action. This attribute is mandatory. You can pick any name you want. This name will be referred to in the Forward element.

2. type – this is the fully qualified class name for the action class. This attribute is mandatory. An action class is a class that is derived from the WA.Core.Configurator.DefaultAction class. We will explain more about this class in the next section. This is an example of the fully qualified class name for the action class: MyCompany.SI.Page1Action, MyCompany.SI, Version=1.0.0.0, Culture=neutral, PublicKeyToken=451cac61f7ec4225

The following is the attribute for the View element:

1. name – this is the name of the view. This attribute is mandatory. You can pick any name you want. This name will be referred to in the Forward element.

The following are the attributes for the Forward element:

1. name – this is the command name that is routed through the Configurator Framework. This attribute is mandatory. A command is usually invoked because of an action on one of the controls in the page, such as when a user clicks a button.
2. action – this is the name of the next action that will be activated by the Configurator Framework when the Configurator Framework processes the command specified in the name attribute.
3. view – this is the name of the view within the specified action activated by the Configurator Framework when the Configurator Framework processes the command specified in the name attribute. This attribute is optional. If not specified, the Configurator Framework uses the first view in the action.

WA.Core.Configurator.DefaultAction Class

As you can see, the Action element in the Map file is associated with an action class. An action class is the class that renders the user interface in the configurator. The action class also constructs and save the properties of the web part.

An action class is derived from the WA.Core.Configurator.DefaultAction class.

The following are the properties of the WA.Core.Configurator.DefaultAction class that you need to know when writing your action class:

1. Configurator – this property gives you access to the Configurator Framework. The WA.Core.Configurator.DefaultAction class is not derived from System.Web.UI.WebControls.WebControl class. If you need to use the facility of a Web Control, such as Page property, use the Configurator.Page property.

When writing your action class, you must be aware of the following WA.Core.Configurator.DefaultAction class methods:

1. public void OnInit() – you can override this method to represent all the controls and adding them to the Configurator Framework. The Configurator Framework invokes the OnInit method of every action classes specified in the Map file.
2. public void OnLoad() – you can override this method to initialize the value of the controls. The Configurator Framework invokes the OnLoad method of every action classes specified in the Map file.
3. public void OnPreRender() - the Configurator Framework calls the OnPreRender method of the active action. This method is a good place to set the visibility of the controls. The Configurator Framework hides all the controls that are added during OnInit. It is the responsibility of the active action to turn on the visibility of the controls. If the active action does not do this, the control is not visible on the page.

4. public void Perform() - the Configurator Framework calls the Perform method of the active action during postback before it activates the next action specified in the Forward element in the Map file. This method is a good place to cache the properties of the web parts to the Configurator before the control is passed to the next action.
5. public void Render(System.Web.UI.HtmlTextWriter output) - the Configurator Framework calls the Render method of the active action. You can use this method to render the control manually using the HTMLTextWriter object.
6. public void GetHelpUrl() - the method gets URL for the help file. If the help URL is provided, the configurator displays the ? icon in the upper-right-hand corner.
7. public void GetInfo() – the method gets the information that is displayed in the information bar of the Configurator window. This is a good place to create a short explanation of what your action class does, for example, configuring the Catalog property of the SI Web Parts
8. protected string GetPostBackEventReference(string command) – the method obtains a reference to a client-side JavaScript function that causes the page to postback when invoked. During postback, the command is processed by the Configurator Framework, for example, you may call this method for the OnClientClick of a Button control.
9. protected string GetPostBackClientHyperlink(string command) – this method is similar with the GetPostBackEventReference except the resulting JavaScript will be prepended with javascript, for example, you may call this method for the NavigateUrl property of a Hyperlink control.

WA.Core.Configurator.ConfiguratorFramework Class

This class represents the Configuration Framework itself. You can get the reference to the framework by using the Configurator property of your class.

You must keep in mind the following properties when implementing your action class:

1. ActionContext — this property gives you access to the WA.Core.Configurator.ActionContext object. You may use the action context as a temporary cache to pass a value from one action to another. The action context is disposed when the configurator is closed. Use Configurator.ActionContext[KeyName] to store or retrieve a value into and from the cache.
2. SystemInfo — this property gives you access to the WA.Core.Configuration.SystemInfo object. SystemInfo object represents the System element in the SI Configuration File that contains this custom configurator. You can use the SystemInfo object to retrieve the information about the System element, such as its name and the Services that it contains.
3. ActiveView — this property give you access to the information about the active view. You can only use this property in the OnPreRender method to get accurate answer. When do you need to use this property? An example will be to hide or show different controls on the page depending on the view that is currently active.

You must keep in mind the following methods when implementing your action class:

1. public void AddControl(Control ctrl) — the method to add a control to the Configurator Framework. Use this method instead of the Configurator.Controls.Add(Control ctrl) method. Otherwise, your control may not be added properly to the Configurator Framework. Usually, you call this method in the OnInit method of your action class. All controls that are added this way are initially hidden by the Configurator Framework. It is the responsibility of the active action to turn on the visibility of the control in the OnPreRender method of the action class.
2. public void RemoveControl(Control ctrl) — the method to remove a control from the Configurator Framework.
3. public bool IsPostBack(IAction action) — the method to check whether the current request is the result of a postback. This method instead of Configurator.Page.IsPostBack property because the Configurator.Page.IsPostBack property may not give you accurate answer if this action is just activated by the Configurator Framework.

4. Indexer — you may use the indexer of the Configuration Framework to store and retrieve the properties of the Web Part. For example, to retrieve the value of the Catalog property, you can call Configurator[Catalog]. The Configurator Framework will attempt to save all the values that you store using the indexer into Web Part properties by the same name when you call Configurator.NotifyChange method. Therefore, you should not use this to store temporary value that you want to pass to another action. In order to do that, use the ActionContext property instead.
5. public void NotifyDeActivated() — the method to close the configurator without saving the properties of the Web Part. For example, you may call this method when the user clicks the Cancel button in the configurator.
6. public void NotifyChange() — the method to close the configurator and save the properties of the Web Part. For example, you may call this method when the user clicks the Save button in the configurator.

Debugging the Custom DAO Provider

In order to debug the code, build the solution in debug mode and copy the resulting DLL and PDB files into the bin folder under the root folder of your SharePoint application. If you do not see the bin folder there, create one. You can also set the output path of your project to this folder so that the DLL and PDB file will be updated everytime you build your project.

Remember to sign your assembly with a strong name key file.

To find out the physical path of the root folder of your SharePoint application

1. Select **Start | Control Panel | Administrative Tools | Internet Information Services (IIS) Manager**.
2. Expand the node with your computer name.
3. Expand the **Web Sites** folder.
4. Find the node that represents your site. The SharePoint site should contain _layouts, _vti_bin and _wpresources underneath it.
5. Right-click the node and select **Properties**.
6. Select the **Home Directory** tab. The value in the Local path tells you the physical path of your root folder.

You should also change the trust level for your web application to Full while debugging the custom DAO Provider. The trust level is specified in the web.config. Find the trust element in your web.config and change the level to Full as follows:

```
<trust level="Full" originUrl="" />
```

Do not forget to specify the type name of your custom DAO Provider in the SI Configuration File.

To debug your custom DAO Provider

1. Open the SharePoint page that contains the SI Web Parts that executes the operation in your external system.
2. Open your project's solution in Visual Studio.
3. Select **Debug | Attach to Process**.
4. Select **w3wp.exe** in the process list. If w3wp.exe is not listed, make sure that you check Show processes in all sessions. You may see more than one w3wp.exe listed. If you do, select the one with your user name.
5. Click **Attach**.
6. Set some breakpoints in your custom DAO Provider code.

7. Refresh the page that contains the SI Web Parts.

Deploying the Custom DAO Provider

When it is ready to deploy the solution, build your solution in release mode. Remember to sign the assembly with a strong name key file. This ensures the assembly created can be deployed into Global Assembly Cache (GAC).

After the assembly is deployed in the GAC, you can change the trust level for your web application back to WSS_Minimal. This is because the assemblies that are deployed in the GAC are considered trusted and therefore do not need the Full trust level.

There are several ways to deploy your assembly into the GAC:

- By using the gacutil.exe that comes with the .NET Framework SDK. Here is the command: gacutil /i <DLLName>
- By using the Microsoft .NET 2.0 Framework Configuration tool. Its shortcut can be found in the "Start > Administrative Tools".
- By dragging your DLL into the C:\Windows\Assembly folder using the Windows Explorer.

Once the assembly is dropped into the GAC, your class is ready to use in the SI Web Parts.

SI Web Parts and Boomi

Quick Apps for SharePoint 2013 integrates with Boomi, a Cloud-based integration solution that allows you to connect any combination of Cloud, SaaS, or On-Premise applications without software or coding. For more information on Boomi, see www.boomi.com.

By integrating with Boomi, you can connect to cloud applications (such as Salesforce.com) and on-premise applications (such as SQL Server) and expose them as REST SOAP Web Services end points to Quest Web Parts for SharePoint 2013 through System Integration (SI web parts).

While the SOAP end points and REST API URL are both available from Boomi, only the SOAP end points are supported by Quest Web Parts for SharePoint 2013.

Integrating Boomi and SI Web Parts

You can create a Boomi process to be deployed to an atom cloud or local atom. An atom acts as an agent between Boomi and the deployed environment. As long as the SharePoint server has access to the SOAP end points, the SI Web Parts can connect and integrate with the Boomi process. Once your integration processes have been deployed to your atom, the atom will contain all the components required to execute your processes from end to end, including connectors, transformation rules, decision handling, and processing logic.

ATOM TYPE	DESCRIPTION
Local Atom	Installed locally on your computer. If your integration scenario includes connecting to resources or applications behind your firewall, such as a database, file system directory or other on-premise applications, you will have to install the Atom locally. The Atom must be installed on a machine that has access to all the required resources.
Cloud Atom	Using the atom from the cloud. If your integration scenario includes connecting to resources or applications accessible via the Internet, such as web applications and FTP sites, you can choose to use an Atom in the Dell Boomi Atom Cloud. This hosted option provides a "zero-footprint" integration solution, with no software or hardware to be installed because all computing is performed in the Dell Boomi data center.

You can build the Boomi process inside Boomi AtomSphere. For more information, see "Building the Boomi Integration Web Service Process" on page 13.

To integrate Quick Apps SI Web Parts with Boomi process

1. Build Boomi integration web service process. See "Building the Boomi Integration Web Service Process" on page 13.
2. Deploy Boomi integration web service process to local Atom or Atom in the cloud.
3. Configure Quick Apps SI Web Parts for system/service to connect to the SOAP URL using Configuration Editor. See "Configuring SI Web Parts" on page 14.
4. Configure the properties in Quick Apps SI Web Parts using ezEdit. See "Configuring SI Web Parts" on page 14.

Building the Boomi Integration Web Service Process

You can evolve any standard process into a web service process by assigning the Web Services Server connector and resetting the base connector in the process flow. If no synchronous responses (output types) are configured for return back to the client application, then any type of process flow is accepted. You can use the other available connectors licensed in your account to connect to other applications and data sources.

Input/Request

The Web Services Server connector allows you to define a single object and operation type that controls the required structure of the request and response data (specify XML data for SOAP requests), and the endpoint URL for all SOAP or data requests (for QuickApps responses, specify Multiple XML Objects even if a single XML object is returned).

The input type defined in your Web Services Server operation is the source document format for your process flow. For example, if XML data is sent in through a SOAP request, then the XML profile must be defined as the source profile in a map (if you want to translate the data directly into a database). You can consider using the Process Call step to execute a chain of child processes for different document flows and actions. The parent process can control the web service to handle all requests and responses.

Output/Response

The Return Documents step controls the return of documents to its source calling point. During process execution, this step gathers all documents that are intended to reach this step, and then sends the documents back to the client application / web server conversation that was opened during a request.

In standard Dell Boomi AtomSphere connectors, a connection component is paired with the operation component to build a direct link to a desired application or data source. However, for web service publishing, connection settings are controlled at the Atom level, so the same web server can be used for all processes that are deployed to the Atom. The Shared Web Server Settings dialog box allows you to specify URL requirements and to control network and security settings at the account and account user level. You need this URL and security settings later in Quick Apps to define the WSDL URL to connect to the Boomi Web services process.

Configuring SI Web Parts

To configure SI Web Parts using the Configuration Editor

1. Open Configuration Editor and add a remote system.
2. Select **WebServices** as System Type.
3. In the Atom's Shared Web Server Properties settings:

IF THE MINIMUM REQUIRED AUTHENTICATION IS SET TO	THEN
None	leave System properties section in Quick Apps Configuration Editor blank or select **Passthrough**
Basic	specify a Boomi user account as username and security token as password. The Boomi user account and security token can be found in Atom's Shared Web Server properties under User Management tab. You may need to click **Generate** and select the **Show** check box to view and copy the security token.

4. From the System Node, add a Service node for the Boomi web services.
5. Enter the WSDL URL and select SOAP as the protocol.

   The WSDL URL is the Base URL (can be found under User Management or Advanced tab and especially if user choose to override the base URL with the appendix "/ws/soap?wsdl").

For more information on the Configuration Editor, see the Configuration Editor User Guide or online help.

To configure SI Web Parts using ezEdit

1. From the SI Web Part, open **ezEdit**.
2. From the Catalog section, select the System and Service defined in the Configuration Editor.
3. Click **Add Operation**, and click **Edit**.

   A list of available Web services methods from Boomi web services processes are listed in the Method drop down list.
4. Select a method.

   Parameters may be displayed.
5. If you need to specify parameters, click **Add Parameter,** and click **Edit**.

6. For the Source, select XML or ComplexType.

   If you specify XML, you must manually create XML strings. If you specify ComplexType, the object graph is shown and you can fill in the property attributes.

FAQs

1. **What basic troubleshooting tasks can I perform if I run into issues?**

   You can view log files to help you troubleshoot issues. Once a process is deployed, go to Manage tab and select Atom Management to access log information by clicking Download logs (including bin). If the Atom is on-premise (vs Atom Cloud), you can directly access the log files under the installed directory of Atom (Program Files/Boomi Atomsphere/Atom-server-instance/logs).

   If you checked everything but things still do not work the way you expected, you might want to try the followings:

   - Restart the Atom - Service Instance Name service (under services.msc)
   - Reset IIS and delete any Web Services proxy assembly (by default, it is located at C:/Windows/Temp, but is configurable through the Configuration Editor)
   - Redeploy the processes and check the log through Atom Management to ensure deployment is successful. You may have to detach an environment, then attach it again for deployment
   - If you are using the SoapUI tool for Web Services testing and have changed WSDL or Boomi Web Services, recreate the test project and point to the WSDL URL. The "Update Definition" menu may not work and many tests will be cached.
   - Check SharePoint ULS log for related log entries for troubleshooting

2. **I am getting errors with the DateTime data type. What should I do?**

   When connecting to Boomi using SOAP, DateTime data type can cause an issue. The date time field in Boomi is by default in the long string format, such as "19980601 000000.000", but an actual database Date type field can be returned. Quick Apps SI Web Parts does not provide option to specify Date Time format string. However from Boomi side, you can create an XML profile for the response. You could specify the field Format Options as a Date/Time field with "MM/dd/yyyy" format. In rare case, you can also involve conversion function when mapping from one profile (such as Database profile) to another (such as XML response profile).

   When updating or creating data from Quick Apps to Boomi, there can also be DateTime field issue. The SI Web Parts Web Services create call parameters using XmlSerializer to deserialize them into DateTime field, and it does not accept the MM/dd/yyyy date string. You can get the error: "The string '' is not a valid AllXsd value". As a workaround, the SI Catalog XML property schema allows for a "DisplayFormat" attribute for the call parameter. You manually enter a format string for the DateTime field you defined as part of the XML element, such as DisplayFormat="{0:yyyy-MM-dd}" or DisplayFormat="{0:yyyy-MM-ddTHH:mm:ssZ}" which will allow XmlSerializer to create the call parameter properly.

3. **I cannot connect using Internet Explorer versions 9 or 10.**

   Those versions of Internet Explorer can cause issues while troubleshooting. Use Firefox or Chrome.

   There is an issue using Internet Explorer in Boomi Atomsphere when designing the process and connecting different shapes on the design surface. The shape can appear to be connected but is actually not. You can drag the shape to see if the connecting line is moving along with the shape, if the connecting line is not moving, t is not connected. Use Firefox instead if this problem persists in Internet Explorer.

4. **I am using Boomi Web Services for the first time to surface SQL Server data, and I am getting an error that is unclear.**

When using Boomi Web Services to surface SQL Server data, you can encounter error. For example, you use the same XML request profile to create or update a Product table, and you get an error indicating "HttpException(500,First document failure: Invalid object name 'Product'.,com.boomi.process.ProcessException: First document failure: Invalid object name 'Product'.)" It appears that this error is thrown from Java servlet, indicating that it is either error at the Web server connector or mapping request to database statement as "product". The error comes from the operation for database update or create using dynamic update/create offered by Boomi. You must qualify your database table with the schema name instead of just "Product", such as "Production.Product".

5. **How do I design Boomi web services process for synchronous call?**

   To send a synchronous response back to the client application (in addition to a standard "200 OK" from the web server), you must understand the Return Documents step. The step supports Web Services Server operation requests made through both URL path definitions and all output types with the exception of "None". In most cases, XML is the expected document format that will reach the Return Documents step, especially if you are responding to a SOAP request. The documents, that reach this step, must match the output type defined in the Web Services Server operation. If no documents reach the Return Documents step, then the message response back to the client application will be blank.

   For example, you want to publish a web service process that listens for SOAP requests to update a local database with customer information based on a company name. Upon successful update of the database records, you want AtomSphere to return the updated customer internal database IDs and a timestamp. You would want to translate any database queries made after the update into an XML format defined in the Output Type field and forward those documents to the Return Documents step.

6. **How do I design the Boomi web services process for pagination when a large result set is returned?**

   Implementing paging to limit the amount of response data sent back to the client application is controlled at the web-server level. Refer to the Boomi AtomSphere online documentation on Web Service and Multiple XML Objects topics.

About Quest Software

Established in 1987, Quest Software (Nasdaq: QSFT) provides simple and innovative IT management solutions that enable more than 100,000 global customers to save time and money across physical and virtual environments. Quest products solve complex IT challenges ranging from database management, data protection, identity and access management, monitoring, user workspace management to Windows management. For more information, visit www.quest.com.

Contacting Quest Software

Email	info@quest.com
Mail	Quest Software, Inc. World Headquarters 5 Polaris Way Aliso Viejo, CA 92656 USA
Web site	www.quest.com

Please refer to our Web site for regional and international office information.

Contacting Quest Support

Quest Support is available to customers who have a trial version of a Quest product or who have purchased a Quest product and have a valid maintenance contract. Quest Support provides unlimited 24x7 access to our Support Portal. Visit our Support Portal at http://www.quest.com/support.

From our Support Portal, you can do the following:

- Retrieve thousands of solutions from our Knowledge Base
- Download the latest releases and service packs
- Create, update, and review Support cases

View the Global Support Guide for a detailed explanation of support programs, online services, contact information, policies, and procedures.

The guide is available at http://www.quest.com/support.

Third Party Components

Quick Apps for SharePoint 2013 contains some third party components (listed below). Copies of their licenses may be found on our website at http://www.quest.com/legal/third-party-licenses.aspx.

COMPONENT	LICENSE OR ACKNOWLEDGEMENT
async.js n/a	Caolan McMahon