TRAINING GUIDE

Lucity Rest API

Lucity Web Services APIs

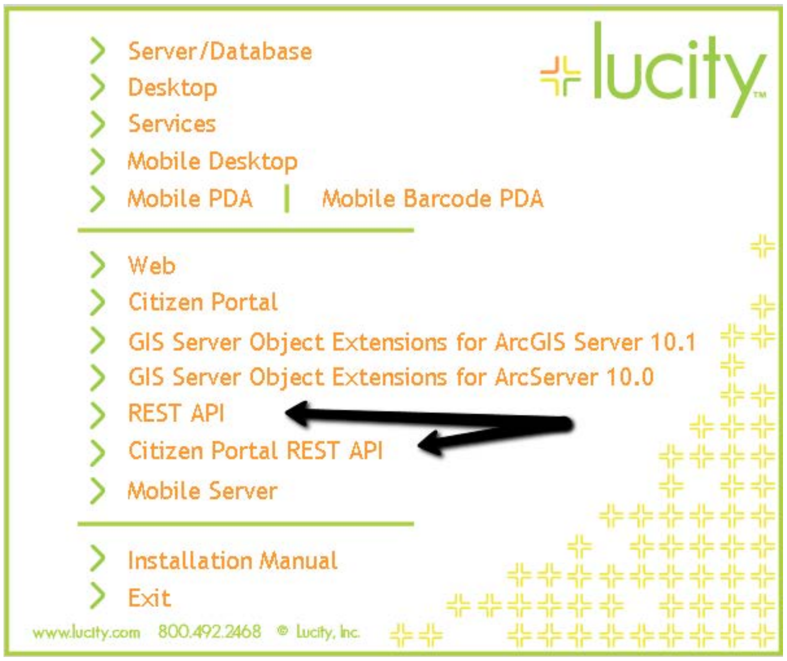
Lucity offers several web service APIs. This guide covers the Lucity Citizen Portal API as well as the Lucity REST API.

Contents

Configuration and Installation

Installation

Both REST APIs are available on the standard install media



The Citizen REST API is designed for customer facing applications and requires no authentication. It contains features such as:

- Creating new requests
- Viewing Existing Requests
- Blocking view of Personally Identifiable Information (PII). It will not serve up any data that has been assigned as PII data. By default this includes things like requestors name, requestors address, requestors email.
- Only serves data that has been marked publically accessible. Every request in Lucity has a flag that identifies whether it should be available for public query. By default all requests that were submitted through the Lucity Citizen Portal REST API have this flag turned on.
- Can auto-reproject coordinates to and from your agency's operational spatial reference (useful when getting data submitted from 311 systems which use Bing or Google or other maps)

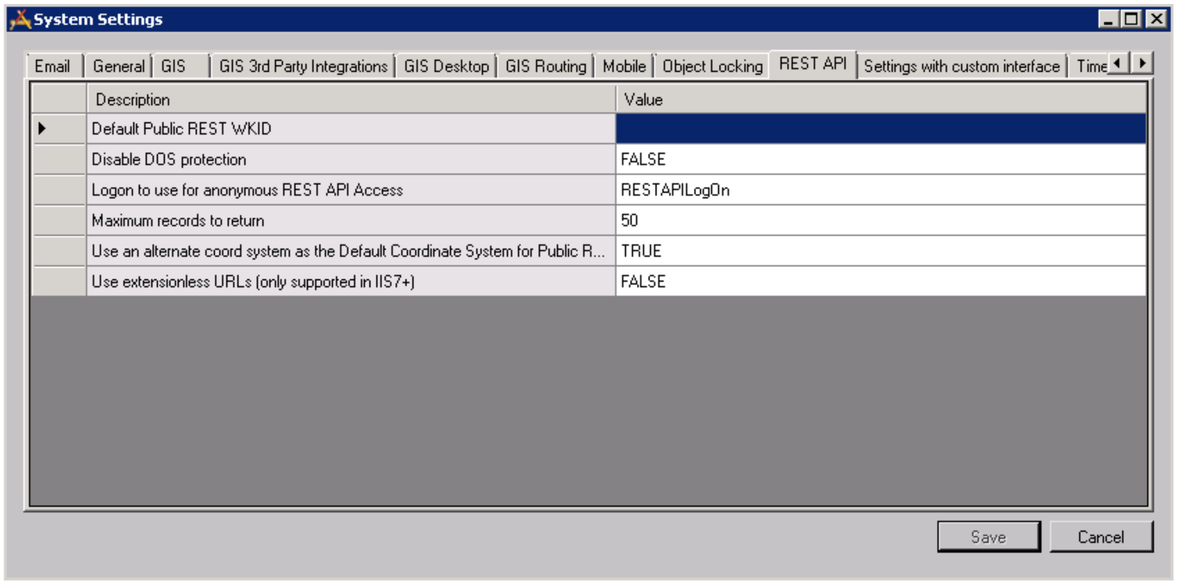
The REST API is designed for internal use applications (GIS, financial integrations, etc.). It requires authentication and used Basic authentication to provide the credentials to the server. For this reason we recommend SSL with this application.

- Supports viewing data from many different Lucity modules
- Supports creating data in many different Lucity modules
- Supports updating data from many different Lucity modules
- Supports deleting data from many different Lucity modules
- Adheres to the user's security profile. If a user is not allowed to delete a work order in Lucity, then they cannot delete the work order in the REST API either.

- Supports a wide range of meta data and extra endpoints that may be useful beyond simple Lucity modules such as :
    - Filters
    - Field Meta Data
    - GIS Configuration Data

Configuration

Several Configuration options are available through System Settings



Description	Value
Default Public REST WKID	
Disable DOS protection	FALSE
Logon to use for anonymous REST API Access	RESTAPILogOn
Maximum records to return	50
Use an alternate coord system as the Default Coordinate System for Public R...	TRUE
Use extensionless URLs (only supported in IIS7+)	FALSE

The Default Public REST WKID allows you to specify that all requests coming in through the Citizen Portal REST API should be assumed to be in a certain coordinate system (for example, Mercator). ESRI documentation provides a full list of available WKID values. This system setting goes hand in hand with the "Use an alternate coord system as the Default Coordinate System for Public Requests" setting. Alternately, if some clients are inserting data using Mercator and some clients are using an alternate coordinate system, the client can include criteria in each call that tells the REST API what the coordinate system is. To do this, include a query param COORDSYS= MERCATOR or COORDSYS=LOCAL.

By default, the REST APIs provide rudimentary denial of service (DOS) protection. If a single IP address makes more than 1000 requests in 10 minutes, the system will not accept requests from that IP address. The number of requests and the number of minutes is also customizable but must be done in the appSettings.config file on the server. The names of the app settings that are applicable are
`DOSREQUESTS`
and `DOSPERIOD`. The latter value is in minutes. To disable the DOS protection for the REST API, change the Disable DOS protection setting to TRUE in the Lucity System Settings.

Maximum records to return limits the total number of records that may be returned on any one REST call. By default, 10 records are returned, but the client can request more records. This system setting sets a cap on how many total records can be returned in one call.
Use extensionless URLs allow prettier URLs if that is something you care about. By default URLs will be like this:

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/55555/TaskList/
With this option set to TRUE the URL will look like this:
http://restapi.lucity.net/gbaMS/Work/WorkOrders/55555/TaskList

Basics

The Lucity REST API uses GET, PUT, POST, and DELETE to perform a variety of actions on resources in Lucity. A resource can be a work order, it can be a list of work orders, it can be a work order task, it can be a hydrant, etc.

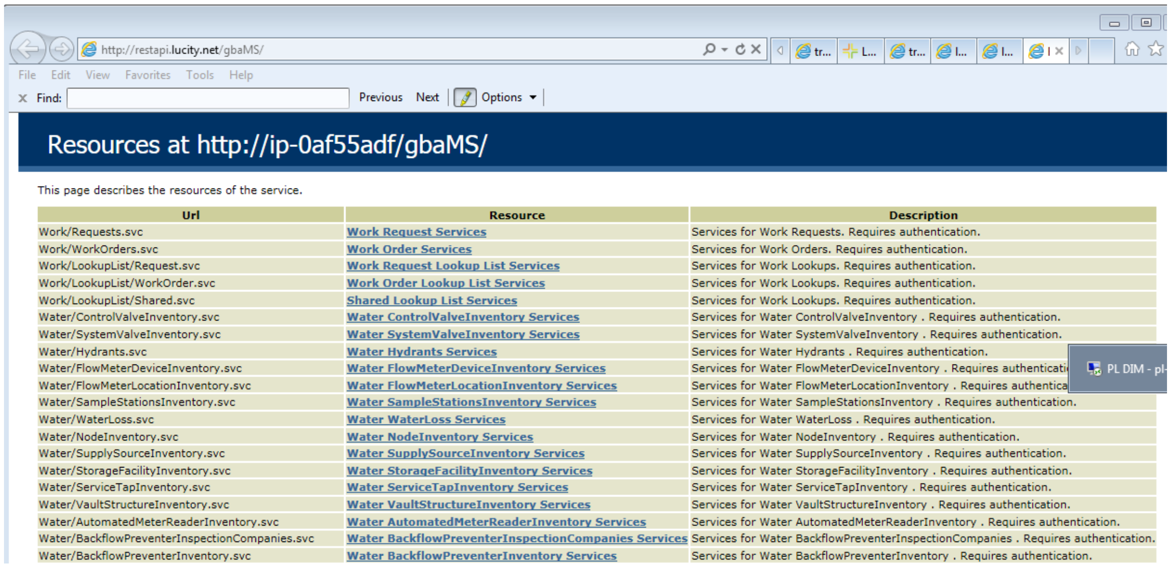
The URL for the web services is composed of the server where it is installed (potentially an alias instead) and the virtual directory (also can be aliased).

http://server/virdir/

Or for example:

http://restapi.lucity.net/gbaMS/ (this is a live functioning link)

The entry page includes a list of all of the available root level resources. There are LOTS of them:


http://restapi.lucity.net/gbaMS/
File Edit View Favorites Tools Help
Find:
Previous
Next
Options
PL DIM - pl

Resources at http://ip-0af55adf/gbaMS/

This page describes the resources of the service.

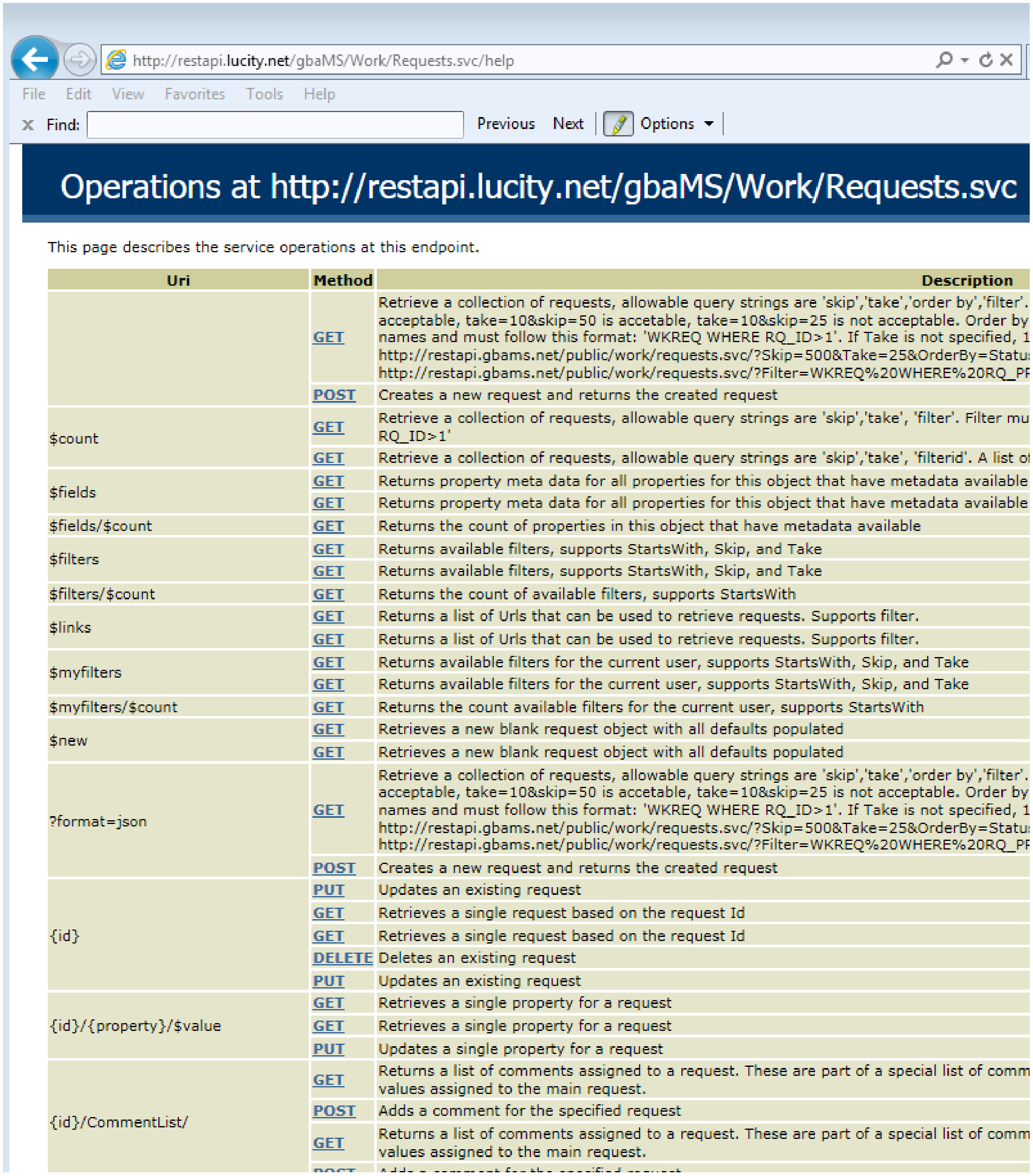
Url	Resource	Description
Work/Requests.svc	Work Request Services	Services for Work Requests. Requires authentication.
Work/WorkOrders.svc	Work Order Services	Services for Work Orders. Requires authentication.
Work/LookupList/Request.svc	Work Request Lookup List Services	Services for Work Lookups. Requires authentication.
Work/LookupList/WorkOrder.svc	Work Order Lookup List Services	Services for Work Lookups. Requires authentication.
Work/LookupList/Shared.svc	Shared Lookup List Services	Services for Work Lookups. Requires authentication.
Water/ControlValveInventory.svc	Water ControlValveInventory Services	Services for Water ControlValveInventory . Requires authentication.
Water/SystemValveInventory.svc	Water SystemValveInventory Services	Services for Water SystemValveInventory . Requires authentication.
Water/Hydrants.svc	Water Hydrants Services	Services for Water Hydrants . Requires authentication.
Water/FlowMeterDeviceInventory.svc	Water FlowMeterDeviceInventory Services	Services for Water FlowMeterDeviceInventory . Requires authenticati
Water/FlowMeterLocationInventory.svc	Water FlowMeterLocationInventory Services	Services for Water FlowMeterLocationInventory . Requires authentica
Water/SampleStationsInventory.svc	Water SampleStationsInventory Services	Services for Water SampleStationsInventory . Requires authentication.
Water/WaterLoss.svc	Water WaterLoss Services	Services for Water WaterLoss . Requires authentication.
Water/NodeInventory.svc	Water NodeInventory Services	Services for Water NodeInventory . Requires authentication.
Water/SupplySourceInventory.svc	Water SupplySourceInventory Services	Services for Water SupplySourceInventory . Requires authentication.
Water/StorageFacilityInventory.svc	Water StorageFacilityInventory Services	Services for Water StorageFacilityInventory . Requires authentication.
Water/ServiceTapInventory.svc	Water ServiceTapInventory Services	Services for Water ServiceTapInventory . Requires authentication.
Water/VaultStructureInventory.svc	Water VaultStructureInventory Services	Services for Water VaultStructureInventory . Requires authentication.
Water/AutomatedMeterReaderInventory.svc	Water AutomatedMeterReaderInventory Services	Services for Water AutomatedMeterReaderInventory . Requires authentication.
Water/BackflowPreventerInspectionCompanies.svc	Water BackflowPreventerInspectionCompanies Services	Services for Water BackflowPreventerInspectionCompanies . Requires authentication.
Water/BackflowPreventerInventory.svc	Water BackflowPreventerInventory Services	Services for Water BackflowPreventerInventory . Requires authentication.

The list goes on.

Note:

Drilling into one of these resources will provide details on what can be done with that resource.

Drilling into the individual help requires authentication, so the browser will prompt for a user name and password. This is your Lucity user name and password. A small snippet of the available actions is shown here:


http://restapi.lucity.net/gbaMS/Work/Requests.svc/help
File Edit View Favorites Tools Help
Find: Previous Next Options

Operations at http://restapi.lucity.net/gbaMS/Work/Requests.svc

This page describes the service operations at this endpoint.

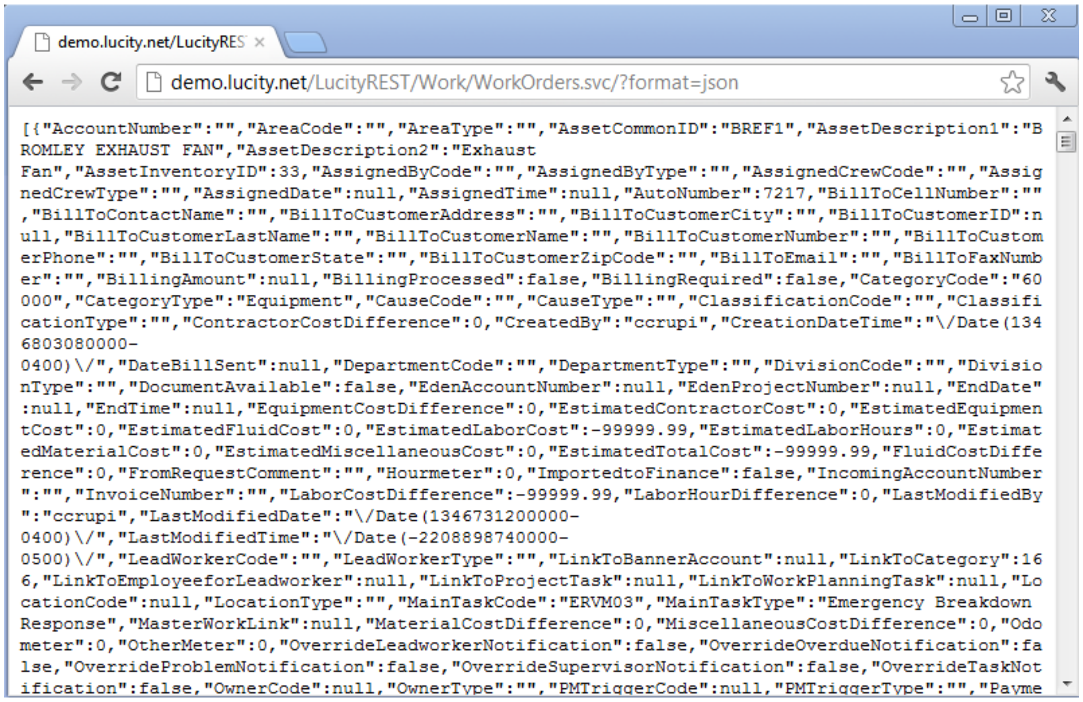
Uri	Method	Description
	GET	Retrieve a collection of requests, allowable query strings are 'skip','take','order by','filter'. acceptable, take=10&skip=50 is accetable, take=10&skip=25 is not acceptable. Order by names and must follow this format: 'WKREQ WHERE RQ_ID>1'. If Take is not specified, 1 http://restapi.gbams.net/public/work/requests.svc/?Skip=500&Take=25&OrderBy=Statu http://restapi.gbams.net/public/work/requests.svc/?Filter=WKREQ%20WHERE%20RQ_PI
	POST	Creates a new request and returns the created request
$count	GET	Retrieve a collection of requests, allowable query strings are 'skip','take', 'filter'. Filter mu RQ_ID>1'
	GET	Retrieve a collection of requests, allowable query strings are 'skip','take', 'filterid'. A list o
$fields	GET	Returns property meta data for all properties for this object that have metadata available
	GET	Returns property meta data for all properties for this object that have metadata available
$fields/$count	GET	Returns the count of properties in this object that have metadata available
$filters	GET	Returns available filters, supports StartsWith, Skip, and Take
	GET	Returns available filters, supports StartsWith, Skip, and Take
$filters/$count	GET	Returns the count of available filters, supports StartsWith
$links	GET	Returns a list of Urls that can be used to retrieve requests. Supports filter.
	GET	Returns a list of Urls that can be used to retrieve requests. Supports filter.
$myfilters	GET	Returns available filters for the current user, supports StartsWith, Skip, and Take
	GET	Returns available filters for the current user, supports StartsWith, Skip, and Take
$myfilters/$count	GET	Returns the count available filters for the current user, supports StartsWith
$new	GET	Retrieves a new blank request object with all defaults populated
	GET	Retrieves a new blank request object with all defaults populated
?format=json	GET	Retrieve a collection of requests, allowable query strings are 'skip','take','order by','filter'. acceptable, take=10&skip=50 is accetable, take=10&skip=25 is not acceptable. Order by names and must follow this format: 'WKREQ WHERE RQ_ID>1'. If Take is not specified, 1 http://restapi.gbams.net/public/work/requests.svc/?Skip=500&Take=25&OrderBy=Statu http://restapi.gbams.net/public/work/requests.svc/?Filter=WKREQ%20WHERE%20RQ_PI
	POST	Creates a new request and returns the created request
{id}	PUT	Updates an existing request
	GET	Retrieves a single request based on the request Id
	GET	Retrieves a single request based on the request Id
	DELETE	Deletes an existing request
	PUT	Updates an existing request
{id}/{property}/$value	GET	Retrieves a single property for a request
	GET	Retrieves a single property for a request
	PUT	Updates a single property for a request
{id}/CommentList/	GET	Returns a list of comments assigned to a request. These are part of a special list of comm values assigned to the main request.
	POST	Adds a comment for the specified request
	GET	Returns a list of comments assigned to a request. These are part of a special list of comm values assigned to the main request.
	POST	Adds a comment for the specified request

Again, there are lots. Drilling into an individual GET link will show what format it is expecting the data in. This may or may not be useful depending on the planned development platform. Developers using .NET should use the serialization objects we provide and not worry about manually writing XML or json.

Working with Data

Getting Data using a browser

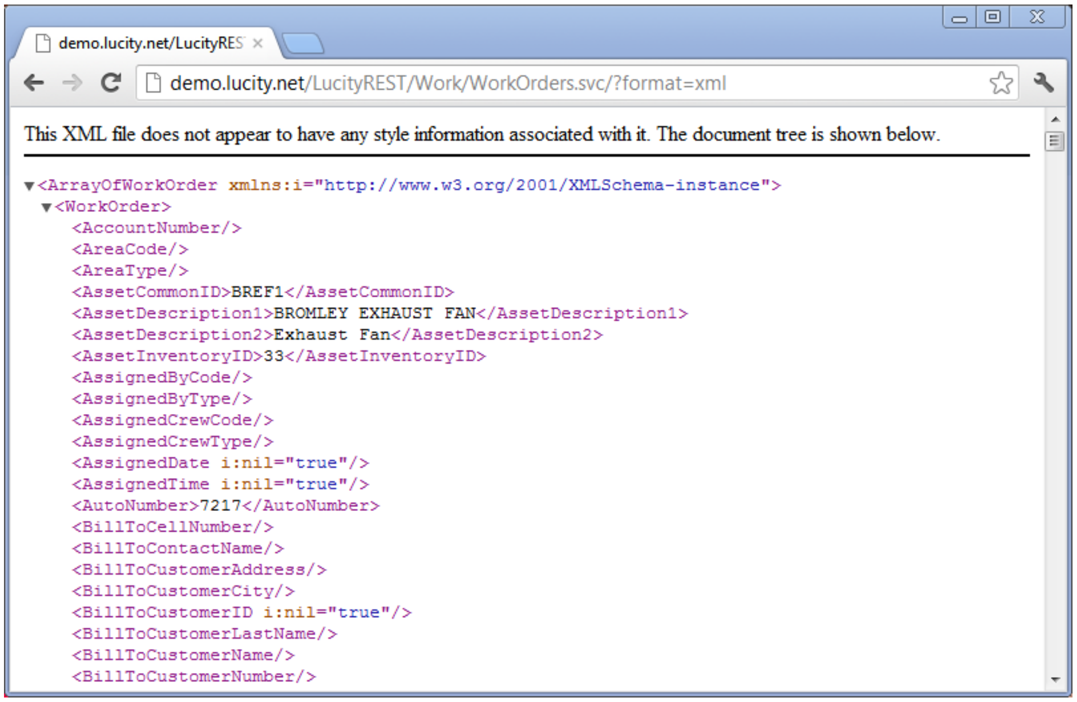
This mostly proves the service can be reached. There probably is no real life application here, but it does provide troubleshooting help.



demo.lucity.net/LucityRES
demo.lucity.net/LucityREST/Work/WorkOrders.svc/?format=json

```
[{"AccountNumber":"","AreaCode":"","AreaType":"","AssetCommonID":"BREF1","AssetDescription1":"B
ROMLEY EXHAUST FAN","AssetDescription2":"Exhaust
Fan","AssetInventoryID":33,"AssignedByCode":"","AssignedByType":"","AssignedCrewCode":"","Assig
nedCrewType":"","AssignedDate":null,"AssignedTime":null,"AutoNumber":7217,"BillToCellNumber":""
,"BillToContactName":"","BillToCustomerAddress":"","BillToCustomerCity":"","BillToCustomerID":n
ull,"BillToCustomerLastName":"","BillToCustomerName":"","BillToCustomerNumber":"","BillToCustom
erPhone":"","BillToCustomerState":"","BillToCustomerZipCode":"","BillToEmail":"","BillToFaxNumb
er":"","BillingAmount":null,"BillingProcessed":false,"BillingRequired":false,"CategoryCode":"60
000","CategoryType":"Equipment","CauseCode":"","CauseType":"","ClassificationCode":"","Classifi
cationType":"","ContractorCostDifference":0,"CreatedBy":"ccrupi","CreationDateTime":"\/Date(134
6803080000-
0400)\/","DateBillSent":null,"DepartmentCode":"","DepartmentType":"","DivisionCode":"","Divisio
nType":"","DocumentAvailable":false,"EdenAccountNumber":null,"EdenProjectNumber":null,"EndDate"
:null,"EndTime":null,"EquipmentCostDifference":0,"EstimatedContractorCost":0,"EstimatedEquipmen
tCost":0,"EstimatedFluidCost":0,"EstimatedLaborCost":-99999.99,"EstimatedLaborHours":0,"Estimat
edMaterialCost":0,"EstimatedMiscellaneousCost":0,"EstimatedTotalCost":-99999.99,"FluidCostDiffe
rence":0,"FromRequestComment":"","Hourmeter":0,"ImportedtoFinance":false,"IncomingAccountNumber
":"","InvoiceNumber":"","LaborCostDifference":-99999.99,"LaborHourDifference":0,"LastModifiedBy
":"ccrupi","LastModifiedDate":"\/Date(1346731200000-
0400)\/","LastModifiedTime":"\/Date(-2208898740000-
0500)\/","LeadWorkerCode":"","LeadWorkerType":"","LinkToBannerAccount":null,"LinkToCategory":16
6,"LinkToEmployeeforLeadworker":null,"LinkToProjectTask":null,"LinkToWorkPlanningTask":null,"Lo
cationCode":null,"LocationType":"","MainTaskCode":"ERVM03","MainTaskType":"Emergency Breakdown
Response","MasterWorkLink":null,"MaterialCostDifference":0,"MiscellaneousCostDifference":0,"Odo
meter":0,"OtherMeter":0,"OverrideLeadworkerNotification":false,"OverrideOverdueNotification":fa
lse,"OverrideProblemNotification":false,"OverrideSupervisorNotification":false,"OverrideTaskNot
ification":false,"OwnerCode":null,"OwnerType":"","PMTriggerCode":null,"PMTriggerType":"","Payme
```

Unfortunately there is no pretty json option, but XML is available which renders better when viewing data like this:



demo.lucity.net/LucityRES
demo.lucity.net/LucityREST/Work/WorkOrders.svc/?format=xml

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ArrayOfWorkOrder xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  ▼<WorkOrder>
     <AccountNumber/>
     <AreaCode/>
     <AreaType/>
     <AssetCommonID>BREF1</AssetCommonID>
     <AssetDescription1>BROMLEY EXHAUST FAN</AssetDescription1>
     <AssetDescription2>Exhaust Fan</AssetDescription2>
     <AssetInventoryID>33</AssetInventoryID>
     <AssignedByCode/>
     <AssignedByType/>
     <AssignedCrewCode/>
     <AssignedCrewType/>
     <AssignedDate i:nil="true"/>
     <AssignedTime i:nil="true"/>
     <AutoNumber>7217</AutoNumber>
     <BillToCellNumber/>
     <BillToContactName/>
     <BillToCustomerAddress/>
     <BillToCustomerCity/>
     <BillToCustomerID i:nil="true"/>
     <BillToCustomerLastName/>
     <BillToCustomerName/>
     <BillToCustomerNumber/>
```

Getting Data using javascript

There are many different methods of getting the data using javascript and a lot of helper libraries out there. These samples use jQuery and one of the lower level functions called .ajax. Because of cross origin concerns, the `jQuery.support.cors` variable is set to true so that jQuery knows that it is okay to send an http request to a server other than the one serving up this web page.

```
<script language="javascript">

        function getdetails()
        {

            //tell jQuery you are going to use cross origin requests (if necessary)
            jQuery.support.cors = true;

            var url = http://restapi.gbams.net/gbaMS/Work/WorkOrders.svc/" + 
$('#woid').val() + "?format=json";

            jQuery.ajax({
                type: "GET",
                url: url,
                username: $('#username').val(),
                password: $('#password').val(),
                success: function (data)
                {
                   $('#wonum').val(data.WorkOrderNumber);
                   $('#woid').val(data.AutoNumber);
                   $('#categorycode').val(data.CategoryCode);
                   $('#actioncode').val(data.MainTaskCode);
                   $('#supervisorcode').val(data.SupervisorCode);

                   getanddisplaycomments(workorderId);

                },
                error: function (XMLHttpRequest, textStatus, errorThrown)
                {
                    if (XMLHttpRequest.status == "404")
                        alert("Work Order does not exist");
                    else if (XMLHttpRequest.status == "401")
                        alert("Invalid credentials");
                    else
                        alert("Error ->" + XMLHttpRequest.status + " " + 
XMLHttpRequest.responseText.Description);
                }
            });

        }

  </script>
```

Posting Data (creating data) using javascript

```
<script language="javascript">

        function addComment(workorderId)
        {

            var params = "{\"Comment\":\"" + $('#comments').val() + "\"}";
            var url = http://restapi.gbams.net/gbaMS/Work/WorkOrders.svc/" + workorderId
+ "/CommentList/?format=json";

            jQuery.ajax({
                type: "POST",
                url: url,
                username: $('#username').val(),
                password: $('#password').val(),
                data: params,
                contentType: "application/json",
                success: function (data)
                {
                    getanddisplaycomments(workorderId);
                },
                error: function (XMLHttpRequest, textStatus, errorThrown)
                {
                    handleError(XMLHttpRequest);
                }
            });

        }

  </script>
```

Notes:____

Putting Data (updating data) using javascript

```
<script language="javascript">

        function update()
        {

            var params = "{\"CategoryCode\":\""
                    + $('#categorycode').val()
                    + "\",\"MainTaskCode\":\""
                    + $('#actioncode').val()
                    + "\",\"SupervisorCode\":\""
                    + $('#supervisorcode').val() + "\"}";

            var url = urlmain + "WorkOrders.svc/" + $('#woid').val() + "?format=json";

            jQuery.ajax({
                type: "PUT",
                url: url,
                username: $('#username').val(),
                password: $('#password').val(),
                data: params,
                contentType: "application/json",
                success: function (data)
                {
                    $('#woid').val(data.AutoNumber)
                    getdetails();
                },
                error: function (XMLHttpRequest, textStatus, errorThrown)
                {
                    handleError(XMLHttpRequest);
                }
            });

        }
  </script>
```

Notes:____

Getting Data using C#

There are also several ways to make HTTP requests with C#. The following gets a list of requests and outputs it to the console window (it will show up as a mess of XML text). It will only return the 10 most recent requests. There is a very similar sample to the below in the ConsoleApplication sample in the DesktopSamples solution.

```
var url = "http://restapi.lucity.net/Public/Work/Requests.svc/";

using (HttpClient client = new HttpClient())
{
        using (HttpRequestMessage request = new HttpRequestMessage(method, url))
        {
                request.Headers.Accept.Add("application/xml");

                using (HttpResponseMessage response = client.Send(request))
                {
                        if (response.Content.HasLength() && response.Content.GetLength() >
0)
                        {
                                using (StreamReader reader = new
StreamReader(response.Content.ReadAsStream()))
                                {
        Console.WriteLine(reader.ReadToEnd());
                                }
                        }
                }
        }
}
```

This is not ideal for working with the data, so a better way is to deserialize the XML into an object that you can work with. We can provide these basic objects to make it easier to work with the data client side. Here is an example partial object:

```
        public class WorkOrder
        {
                public string AccountNumber { get; set; }
                public string AreaCode { get; set; }
                public string AreaType { get; set; }
                public int? AssetInventoryID { get; set; }
                public string AssignedByCode { get; set; }
                public string AssignedByType { get; set; }
                public string AssignedCrewCode { get; set; }
                public string AssignedCrewType { get; set; }
                public DateTime? AssignedDate { get; set; }
                public DateTime? AssignedTime { get; set; }
                public int? AutoNumber { get; set; } ……
        }
        public class ArrayOfWorkOrder: List<WorkOrder>
        {
        }
```

The objects we provide also include some attributes that assist with serialization not shown in the above clip. They are included in a series of C# projects (or compiled dlls) called Lucity.*.SerializationObjects.

These objects are current as of whenever you downloaded them from our site. It is not necessary to update these client side objects with each release. You only need to update the objects if there is a new property you want to take advantage of in the new release.

The following example uses WebClient instead of HttpClient and deserializes the data into a collection object and then binds a grid on a windows form to the collection. This is part of SimpleWindowsFormsWorkOrderSample in the DesktopSamples solution.

```
            WebClient client = new WebClient();
            client.UseDefaultCredentials = false;
            client.Credentials = new NetworkCredential(textBoxUserName.Text, 
textBoxPW.Text);

            string data = null;
            try
            {
                data = 
client.DownloadString("http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/");
            }
            catch (WebException ex)
            {
                MessageBox.Show(ex.Message);
            }

            System.Xml.Serialization.XmlSerializer xml = new 
System.Xml.Serialization.XmlSerializer(typeof(ArrayOfWorkOrder));
            dataGridView1.DataSource  = (ArrayOfWorkOrder)xml.Deserialize(new 
System.IO.StringReader(data));
```

Notes:____

Posting (Creating) Data using C#

This example creates a Request object (available in the Lucity.Work.Serialization project or assembly) and serializes it posting the data to the server to create a new request. The results of the operation are written to a list box called Results. This sample is part of SimpleWindowsFormsWorkOrderSample in the DesktopSamples solution.

```
            Request complaint = new Request();
            complaint.ProblemCode = comboBoxProblem.SelectedValue.ToString();

            //creates a new request using some helper methods
            XmlSerializer ser = new XmlSerializer(typeof(Request));
            System.IO.MemoryStream stream = new System.IO.MemoryStream();
            System.Xml.XmlTextWriter xmlWriter = new System.Xml.XmlTextWriter(stream,
Encoding.UTF8);
            ser.Serialize(xmlWriter, complaint);
            xmlWriter.Flush();
            stream.Seek(0, System.IO.SeekOrigin.Begin);


            WebClient client = new WebClient();
            client.Headers.Add("Content-Type", "text/xml");
            client.Encoding = System.Text.Encoding.UTF8;
            client.UseDefaultCredentials = false;
            client.Credentials = new NetworkCredential(textBoxUserName.Text,
textBoxPW.Text);

            try
            {
                string dataForUpload = Encoding.UTF8.GetString(stream.ToArray());
                string result = client.UploadString(new
Uri("http://restapi.lucity.net/gbaMS/Work/Requests.svc/"), "POST", dataForUpload);
                Request req = (Request)ser.Deserialize(new
System.IO.StringReader(result));

                Results.Items.Add(String.Format("New Request Created: {0}, Number: {1}",
req.AutoNumber, req.RequestNumber));

            }
            catch (WebException exc)
            {
                MessageBox.Show(string.Format("Failed to submit request. {0}",
exc.Message));
            }
```

There are many more samples including asynchronous options and WPF available in the DesktopSamples Solution.

Notes:____

Key Concepts

Formats (json, XML)

Two formats are supported with the REST APIs: json and xml. Each request must include the format in the query string of the url (the default format is xml, so technically it is only required for json requests).

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/?format=json

Returns data in json

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/?format=xml or

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/

Returns data in xml.

For posting and putting (creating and updating) data, the format must be included both in the query string

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/?format=json

And in the Content-Type of the request header. The content type for json is "application/json" or "text/javascript" and the content type for xml is "text/xml".

Special Requirements for XML

It is not necessary to include all properties on an object when posting. If you are using the Lucity serialization objects and serializing them that will just happen automatically, but if you are manually building XML it is perfectly acceptable to include just the properties you want to update or insert. For example, a request can be inserted with the following body:

<Request><ProblemCode>test</ProblemCode></Request>

Which will insert a new request with the problem code = test.

What is required, however, is that whatever properties are included are listed in alphabetical order. Again, if you are using the Lucity Serialization objects this is irrelevant.

Errors

The Lucity REST API tries to follow the HTTP standard for errors. The following is a table of errors that could be returned:

401 UNAUTHORIZED	The user likely did not provide credentials. The Lucity REST API will issue a challenge which will cause most browsers (if the client is a javascript client) to prompt for username and password. If you get this prompt, most likely you did not provide credentials or did not handle a 401 error from the client.
404 NOT FOUND	The requested record was not found or the client made a request to a URL that does not exist and should consult the help listing for available endpoints. For example http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/help contains the available endpoints for Work Orders.

400 BAD REQUEST	The client application requested something that the server considers invalid. This could be a problem in the formatting of the data sent to the server, it could be that the data sent to the server causes a rules violation (like an invalid problem code). 400 Errors will generally include a description of the problem.
500	This is a server error. Check the logs on the server for the reason for the error. Typically these errors will show up both in the rolling.log and in the event viewer on the web server.
412 PRECONDITION FAILED	On a PUT or DELETE, if the client includes an If-Match header and the ETAG does not match with the current version in the database, this error will be returned. This protects from overwriting another user's data, but is completely optional and the client system is responsible for including the header if this behavior is desired.
405 METHOD NOT ALLOWED	This is returned if the client has requested to do something we do not support. For example, if a user tries to delete a request using the Lucity Citizen Portal REST API.

Expected return codes and behavior

Return status codes and headers also follow HTTP standards. The following are the possible return codes you will see developing with the REST API.

200 OK	A GET request will return this code if there were no problems with the request. There will be returned in the body of the response. A PUT will also return this status code if it was successful and will return a copy of the object in the body of the response as it currently exists in the database (this might include data changed after calculations, additional defaults, etc.).
201 CREATED	A POST request that creates a new record will return this status code and will return the copy of the object in the body of the response as it currently exists in the database.
204 NO CONTENT	A DELETE request that successfully deletes a record will return this status code. The body of the response will be empty.
304 NOT MODIFIED	A GET response which includes an ETAG may return a 304 not modified if the object in the database has not been modified since the original ETAG was issued.

Lists

All lists will have a trailing backslash (query parameters go after the slash however). For example:

List of work orders and a list of work orders that are in a designated filter

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/?FilterId=123

List of available code types for MainTaskCode

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/MainTaskCode/

ETags

ETags are returned on all GET that return a single record as well as all PUT and POST requests. The system will respect ETags for GETS, PUTS, and DELETES.

For example, if a GET on a single work order is issued and it includes a header with the following:

```
If-None-Match: "686897696a7c876b7e"
```

The Lucity REST API will check the version of the work order in the database, if it matches the version provided in this ETAG, the Lucity REST API will return a 304 NOT MODIFIED.

For Updates, ETAGS can be used to make sure that one user does not overwrite another user's data. If, on a PUT, the following is included on the header:

```
If-Match: "686897696a7c876b7e"
```

The Lucity REST API will only make the requested update if the version of the record in the database matches the provided ETAG.

Query Parameters

The following query parameters are supported (not all requests support all query parameters)

FILTER	Filter string which should general include a table name. For example: Filter=WKREQ WHERE RQ_STAT_CD < 950 (provided unencoded here for clarity)
FILTERID	Filter Id that should be used to filter the data. Each module has a list of canned or predefined filters that have been saved by users. This is the AutoNumber that represents that saved filter. These filters are available as a list that you can show users (See Special Functions).
ORDERBY	The property or field name that the data should be sorted by. This is the autonumber property by default (descending order). For example: OrderBy=StatusCode+DESC
TAKE	Designates how many records should be returned. By default 10 will be returned. The max is limited based on the Lucity Administrator assigned max in Lucity System Settings (max is defaulted to 50).
SKIP	Designates how many records to skip. For example, to get the second set of 15 records, the Skip = 15 and the Take = 15. Skip MUST be divisible by Take. You cannot request Skip = 20 and Take = 15. This will result in a 400 Bad Request error.
FORMAT	format=json or format=xml. The default is xml.
STARTSWITH	This is a special query parameter only available for certain lookup lists such as category, problem, or standard code type values. For example: STARTSWITH=a will return all code types where the code (and in some cases the type) starts with an a.
COORDSYS	This is a special query parameter only used with the Citizen Portal REST API. See documentation on configuration for a discussion on this parameter. Available options are COORDSYS=LOCAL, COORDSYS=MERCATOR

Special Functions

Getting Record Counts: $count

This keyword will always occur after a trailing slash and can include query parameters. It will return the total record count (with respect to the supplied query parameters).

GET http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$count

Returns a count of all work orders

GET
http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$count?Filter=WKORDER+WHERE+WO_STAT_CD%3C950

Returns a count of all open work orders (where WO_STAT_CD < 950)

GET http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/MainTaskCode/$count

Returns a count of all available code type values for MainTaskCode on a work order.

Getting pick lists: {PropertyName}/ and {PropertyName}/$count

Fields that are pick lists (code types, problem codes, time codes, category codes, etc.) have a special endpoint that can return all of the valid pick list items. The format of the URL is

http://restapi.lucity.net/[Program]/[Module].svc/[Optionally the Child List]/[PropertyName]

For example:

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/PriorityCode/

Return all available priority codes.

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/WorkOrderTaskList/UnitofMeasureCode/

Returns all available unit of measure values for the tasks on work orders. Property names are case sensitive on these requests.

Work flow code types (such as problem, cause, and task) also follow this convention. All available problems which can be assigned to a work order are at this endpoint:

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/MainTaskCode/

It is also possible to get a list of only the items associated to a specific category.

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/CategoryCode/01000/MainTaskCode/

Returns all main task codes associated to a category with category code = 01000

Getting a new empty object: $new

To get a new object with all defaults populated use the $new keyword.

GET http://restapi.lucity.net/gbaMS/Work/Requests.svc/$new

Returns a request object with all of the defaults populated (such as status code and any other custom defined defaults as well). This call makes no changes to the data on the server.

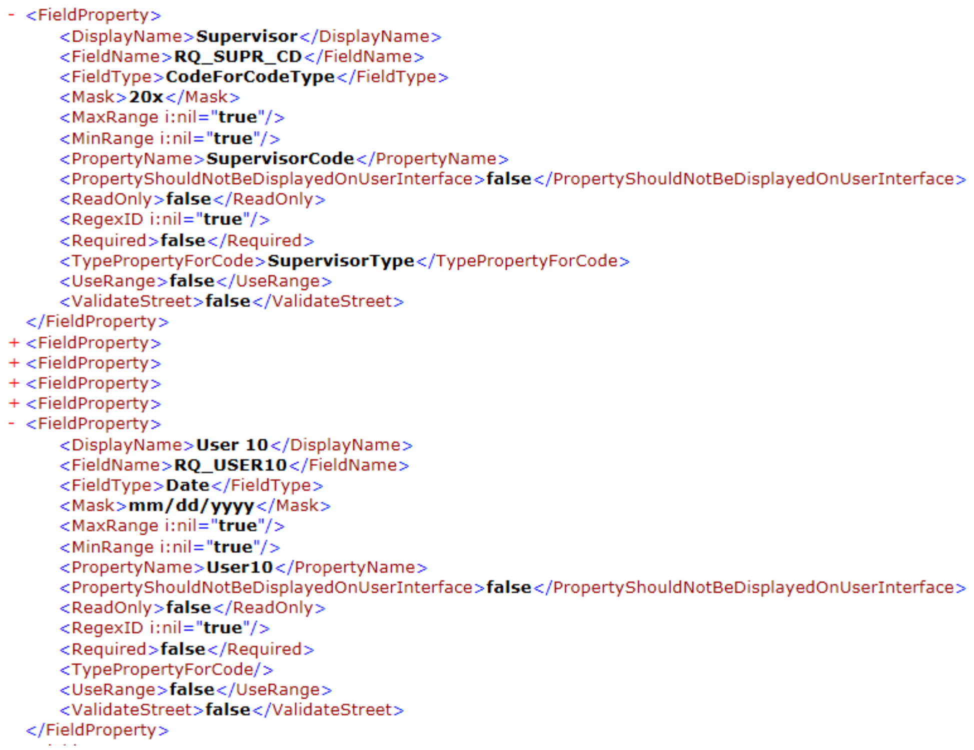
Getting metadata: $fields

To get metadata about fields such as the user defined display name, maximum mask, field type, whether it is required or read-only, use the $fields keyword.

For example:

http://restapi.lucity.net/gbaMS/Work/Requests.svc/$fields

Returns all of the properties and fields on a request and details about each field:



```
- <FieldProperty>
      <DisplayName>Supervisor</DisplayName>
      <FieldName>RQ_SUPR_CD</FieldName>
      <FieldType>CodeForCodeType</FieldType>
      <Mask>20x</Mask>
      <MaxRange i:nil="true"/>
      <MinRange i:nil="true"/>
      <PropertyName>SupervisorCode</PropertyName>
      <PropertyShouldNotBeDisplayedOnUserInterface>false</PropertyShouldNotBeDisplayedOnUserInterface>
      <ReadOnly>false</ReadOnly>
      <RegexID i:nil="true"/>
      <Required>false</Required>
      <TypePropertyForCode>SupervisorType</TypePropertyForCode>
      <UseRange>false</UseRange>
      <ValidateStreet>false</ValidateStreet>
  </FieldProperty>
+ <FieldProperty>
+ <FieldProperty>
+ <FieldProperty>
+ <FieldProperty>
- <FieldProperty>
      <DisplayName>User 10</DisplayName>
      <FieldName>RQ_USER10</FieldName>
      <FieldType>Date</FieldType>
      <Mask>mm/dd/yyyy</Mask>
      <MaxRange i:nil="true"/>
      <MinRange i:nil="true"/>
      <PropertyName>User10</PropertyName>
      <PropertyShouldNotBeDisplayedOnUserInterface>false</PropertyShouldNotBeDisplayedOnUserInterface>
      <ReadOnly>false</ReadOnly>
      <RegexID i:nil="true"/>
      <Required>false</Required>
      <TypePropertyForCode/>
      <UseRange>false</UseRange>
      <ValidateStreet>false</ValidateStreet>
  </FieldProperty>
```

Working with Existing Filters: $filters and $myfilters

To let a user pick from a list of existing filters, use the $filters and $myfilters keyword. This keyword supports query parameters StartsWith, Skip, and Take.

For example:

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$myfilters

Returns all Work Order filters for the currently logged in user

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$filters

Returns all Work Order filters for all users

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$myfilters/$count

Returns a count of all filters available for the currently logged in user

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$myfilters/$count?StartsWith=A

Returns a count of all filters for the currently logged in user that start with A. and the following syntax returns these filters

http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$myfilters?StartsWith=A

Clearing Cached Data: $cache

Lucity caches certain data to make lookups faster and to reduce the load on the database server. This includes things like pick list values, field properties, as well as others.

The REST APIs are stateless web service applications. Restarting the app pool will not upset current users other than causing a very small delay the next time a request is made. However, restarting the app pool is not always realistic and is not possible directly from a client application.

It is possible to force a cache to clear on the REST API server from a client application using the $cache keyword and the DELETE method.

DELETE http://restapi.lucity.net/gbaMS/Work/WorkOrders.svc/$cache

Will clear the caches that are associated with the Work Order objects. This should return a 204 NO CONTENT if the cache clear was successful.

Other Notes

The REST API currently will currently not overwrite valid values with null values (even if that is the desired outcome).

Launching Lucity Applications from another application

While not directly related to the Lucity REST APIs, launching the Lucity desktop or the Lucity Web applications to display a specified record or filter is a common need.

Launching the Lucity Desktop Application from Javascript

Lucity provides an Acitve X control that can be used to launch the Lucity Desktop application using javascript or another language. Only Internet Explorer has built in support for interacting with Active X controls.

There are several pieces of information you need to launch the Lucity desktop software to a specific record from javascript.

a. Client identifier: Lucity uses a concept of client numbers. It is how we distinguish between different sets of databases at a single client site (test, production, development, etc). Usually the client number for the default production client will be “clint001” but in theory it could be “clint002”, “clint003”, etc.

b. User Name: This is the Lucity logon Id that you want used when Lucity opens. It needs to be the logon Id for the current user. This logon Id cannot be shared across multiple users. You would likely need to prompt the user in some way to get this Id. It is the same User Id that is used when authenticating to the Lucity REST API. No password is required here, however.

c. Module Path: This is a string that identifies which module you want to open. It is in the format [Program Name]//[[Module Name]]. There is a comprehensive list of these in the MODULES table in the GBAUser database. The Program Name value comes from the Program_Name field. The Module Name comes from the Module_Name field. Here is the string for Work Orders “Work//WKOrder”

d. filterString: This is the filter that defines what you want to see in the desktop software. So if you want to see one specific work order Id your filter string would be “WKORDER WHERE WO_ID = [the Id you want to see]”. This is basically everything after the “FROM” in a sql statement.

```
function launchLucityClient(filterString, modulePath, client, userName)

{

                document.body.style.cursor = 'wait';

                var lucityClient = new ActiveXObject('GBAIntegEXE.cGBAIntegOutofProcess');

                lucityClient.Initialize(client, userName);

                lucityClient.View(modulePath, filterString);

                document.body.style.cursor = 'default';

}
```

Example usage that opens the Work Order module to a work order id of 1212

```
launchLucityClient( "WKORDER WHERE WO_ID = 1212", "Work//WKOrder",
"clint001","JOE_USER");
```

Launching the Lucity Web Application using a URL

The following information is required to open the web software:

lucityWebUrl: This is the url to the Lucity Web application

moduleId: This is the module Id for the module you want to open. A comprehensive list of module Ids is available in the KeyId field of the Modules table in the GBAUser database. The module Id for work orders is 48.

filterString: This is the filter string for the module you want to open. For example, to open work order 1212 the string would be WKORDER WHERE WO_ID = 1212

tabName: This is a simple string that describes what you are showing. It is the tab name that is used when displaying the data to the user.

This is the url:

[lucityWebUrl]/Public/Routing.aspx?RouteTarget=Internal&RouteSubTarget=Views&RouteAction=OpenDefault&RouteParam1=[moduleId]&RouteParam2=[tabName]&RouteParam3=[filterString]

(*note that the above URL is not shown encoded. You may want to encode the URL)

Example usage that opens work orders to a work order Id of 1212

http://demo.lucity.net/LucityWeb/Public/Routing.aspx?RouteTarget=Internal&RouteSubTarget=Views&RouteAction=OpenDefault&RouteParam1=48&RouteParam2=My+Work+Order&RouteParam3=WKORDER+WHERE+WO_ID%3D1212&end=true