lucity™

TRAINING GUIDE

Lucity Web Services APIs

clarity through connections www.lucity.com 10561 Barkley, Suite 500 Overland Park, KS 66212 913.341.3105

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

How it Works

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/LucityRESTAPI/ (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:

Url	Resource	Description
Work/Requests.svc	Work Request Services	Services for Work Request . Requires authentication.
Work/WorkOrders.svc	Work WorkOrder Services	Services for Work WorkOrder . 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.
GIS/Maps.svc	GIS Map Setup Services	Services for Maps. Requires authentication.
GIS/MapServices.svc	GIS Map Services	Services for Map Services. Requires authentication.
GIS/FeatureLayers.svc	GIS Feature Layers	Feature Layers. Requires authentication.
GIS/Basemaps.svc	GIS Base Map Services	Base Maps. Requires authentication.
Work/LookupList/Shared.svc	Shared Lookup List Services	Requires authentication.
Shared/Streets.svc	Street List Services	Requires authentication.
Shared/SystemSettings.svc	Shared System Setting Services	Services for System Settings. Requires authentication.
System.svc	System Services	Services for system wide actions.
Shared/VersionInfo.svc	Version Information	Services for Version Information. Requires authentication.
Shared/Filters.svc	Filter List Services	Requires authentication.
Work/WorkCategory.svc	Work WorkCategory Services	Services for Work Categories . Requires authentication.
Shared/PropertyQuery.svc	Property Query Services	Requires authentication.

The list goes on.

Notes:_____

_____

_____

_____

_____

_____

_____

_____

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:

Uri	Method	Description
	GET	Retrieve a RequestCollection, allowable query strings are 'skip','take','order by','filter'. If specified, take must be a factor of skip. take=10&skip=0 is acceptable, take=10&skip=50 is accetable, take=10&skip=25 is not acceptable. Order by may be specified using property or field names. Filte must use field names and must follow this format: 'WKORDER WHERE WO_ID>1'
	POST	Creates a new Request and returns the created object
$actionlinks/	GET	Retrieves all available actions
	GET	Retrieves all available actions
$cache	DELETE	Invalidates caches held by the resource
$count	GET	Retrieve a RequestCollection, allowable query strings are 'skip','take', 'filter'. Filter must use field names and must follow this format: 'WKREQ WHERE RQ_ID>1'
$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 for the current user, supports StartsWith
$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 of 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
$nonpersistent	POST	Creates a new Request and does not save it, but returns the object in its state before Save
	POST	Creates a new Request and does not save it, but returns the object in its state before Save
?format=json	GET	Retrieve a RequestCollection, allowable query strings are 'skip','take','order by','filter'. If specified, take must be a factor of skip. take=10&skip=0 is acceptable, take=10&skip=50 is accetable, take=10&skip=25 is not acceptable. Order by may be specified using property or field names. Filte must use field names and must follow this format: 'WKORDER WHERE WO_ID>1'
	POST	Creates a new Request and returns the created object
		Returns a list of comments assigned to a request. These are part of a special list of comments that

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. It makes things much easier.**

Special Functions

Functions to Help and assist your efforts

Starting Point to get a list

http://restapi.lucity.net/LucityRESTAPI/Work/WorkOrders.svc/help

$field – Metadata - A Data Dictionary VERY useful

http://restapi.lucity.net/LucityRESTAPI/Work/WorkOrders.svc/$fields

$actionlinks – provides more urls to use in subsequent work

http://restapi.lucity.net/LucityRESTAPI/Work/WorkOrders.svc/{autoid}/$actionlinks/

$count – just that counts

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

Others to keep in mind

$cache – Invalidate the cache held by that object

$value – get the value of one property – (find property names via $field)

http://restapi.lucity.net/LucityRESTAPI/Work/WorkOrders.svc/34110/CategoryCode/$value

$new – form a new object with defaults

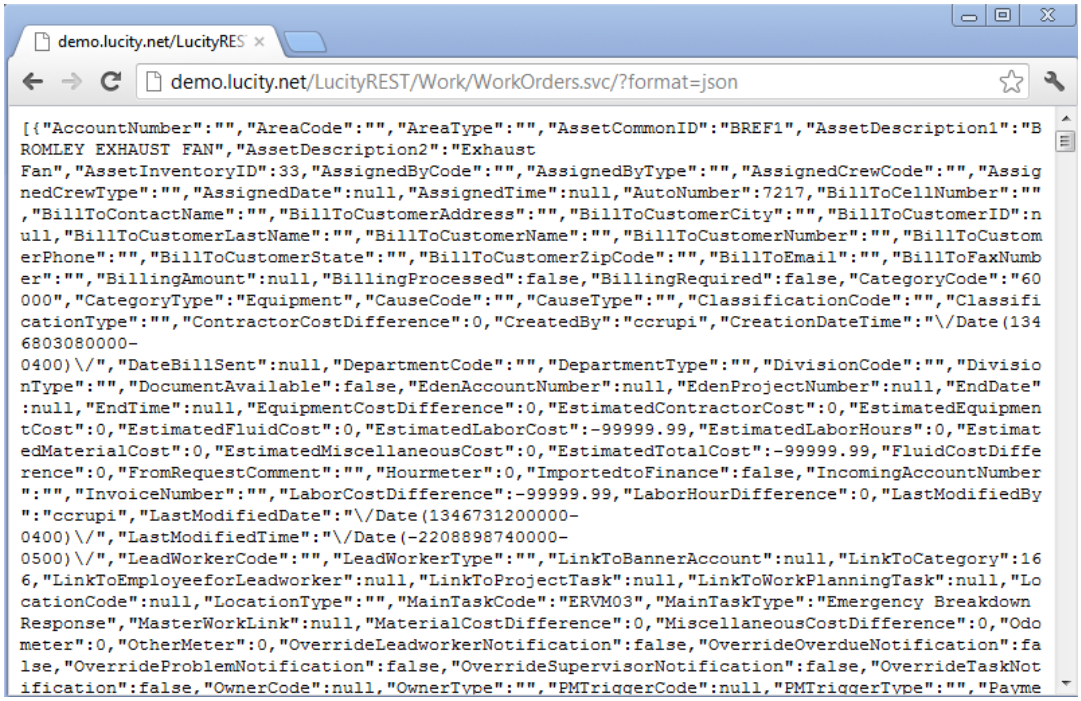
?commonid – supported in many but not all inventory items

http://restapi.lucity.net/LucityRESTAPI/Work/WorkOrders.svc/?commonid=2016-01558

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.

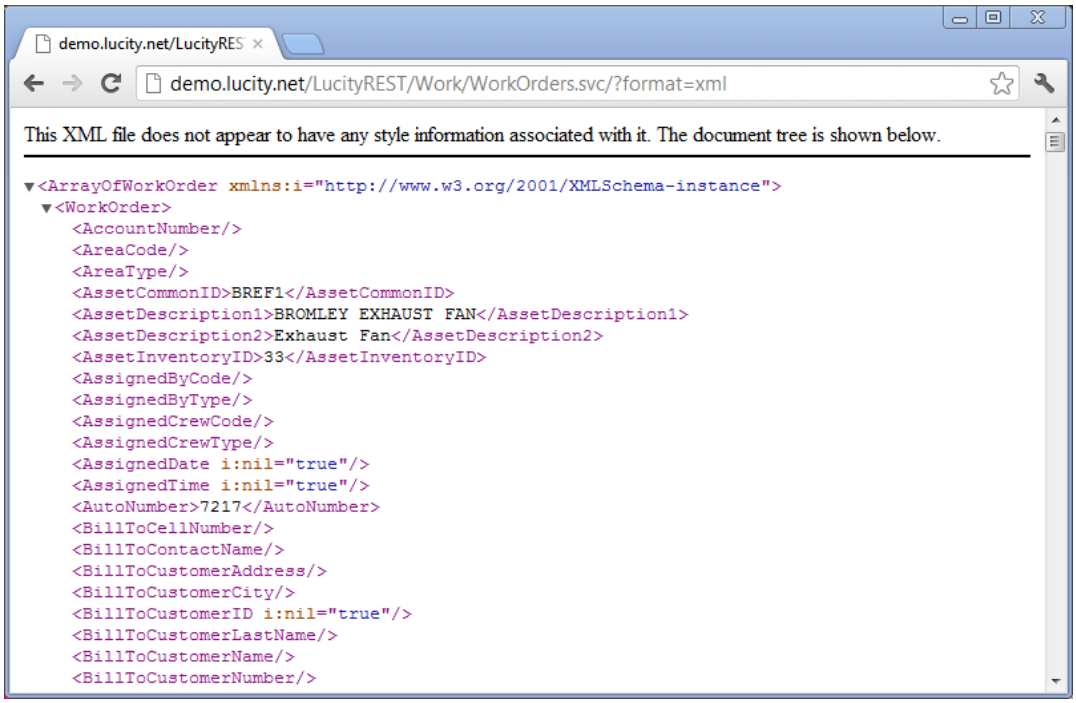


Notes:_____

_____

_____

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



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.

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

    jQuery.ajax({
        type: "GET",
        url: url,
        username: $('#username').val(),
        password: $('#password').val(),
        cache: false,
        success: function (data)
        {
            filloutform(data);
        },
        error: function (XMLHttpRequest, textStatus, errorThrown)
        {
            handleError(XMLHttpRequest);
        }
    });

}
```

Posting Data (creating data) using javascript

```
function create()
{

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

    var url = urlmain + "WorkOrders.svc/?format=json";

    jQuery.ajax({
        type: "POST",
        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);
        }
    });

}
```

Notes:_____

Putting Data (updating data) using 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);
        }
    });

}
```

Notes:_____

_____

_____

_____

_____

_____

_____

_____

Getting Data using C#

There are also several ways to make HTTP requests with C#. The following creates a Request. It has the additional functionality of setting a couple of properties on the Request prior to posting. There is a very similar sample to the below in the Simple Windows Forms Work Order sample in the Desktop Samples solution.

```
{
    Request complaint = new Request();
    complaint.ProblemCode = comboBoxProblem.SelectedValue.ToString();
    complaint.CategoryCode = "02000";

    //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/LucityRESTAPI/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));

        _lastCreatedRequestID = (int)req.AutoNumber;

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

In working with the data, 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 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/LucityRESTAPI/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:_____

_____

_____

_____

_____

_____

_____

_____

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/LucityRESTAPI/Work/WorkOrders.svc/?format=json

Returns data in json

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

http://restapi.lucity.net/LucityRESTAPI/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/LucityRESTAPI/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".

Errors and Codes

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/LucityRESTAPI/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.

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

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/LucityRESTAPI/Work/WorkOrders.svc/PriorityCode/

Return all available priority codes.

http://restapi.lucity.net/LucityRESTAPI/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/LucityRESTAPI/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/LucityRESTAPI/Work/WorkOrders.svc/CategoryCode/10000/MainTaskCode/

Returns all main task codes associated to a category with category code = 10000

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.

Notes:_____

_____

_____

_____

_____

_____

Supported API's

Lucity Citizen Portal REST API:

Targeted at:

- Citizen facing interfaces
- Supports anon access, only serves data flagged as for the public, protects PII, (personable identifiable information) with a more restrictive security (supports GET and POST and only on very limited endpoints)

Lucity REST API:

Targeted at:

- everything else (GIS, financials, applications for employees or possible contractors)
- Users must log in
- Full featured

Licensing

- The REST API is a purchased license component with Lucity
- It is currently a site license purchase
- Support and maintenance on this product provides developer support (one of our developers working with your developer). We do not provide end user support for products developed with our API
- It is a separately installed web application

Resources

- GitHub repository
  - https://github.com/LucityInc/lucity-restapi-samples
- API help guide
- Service directory

Notes:_____

_____

_____

_____

_____