Sage SalesLogix | White Paper

SData Endpoints
An Introduction to SData Endpoints in Sage SalesLogix v7.5.4
An Introduction to SData Endpoints

Documentation Comments
This documentation was developed by Sage SalesLogix User Assistance. For content revisions, questions, or comments, contact the Sage SalesLogix writers at saleslogix.techpubs@sage.com.

Copyright
Copyright © 1997-2011, Sage Software, Inc. All rights reserved. This product and related documentation are protected by copyright and are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of this product or related documentation may be reproduced in any form by any means without prior written authorization of Sage and its licensors, if any.

Version
Version 7.5.4

Trademarks
Sage SalesLogix is a registered trademark of Sage Software, Inc. Sage, the Sage logos, SalesLogix, and the Sage product and service names mentioned herein are registered trademarks or trademarks of Sage Software, Inc., or its affiliated entities. All other trademarks are the property of their respective owners.

Disclaimer
Sage has thoroughly reviewed this manual. All statements, technical information, and recommendations in this manual and in any guides or related documents are believed reliable, but the accuracy and completeness thereof are not guaranteed or warranted, and they are not intended to be, nor should they be understood to be, representations or warranties concerning the products described. Sage assumes no responsibility or liability for errors or inaccuracies with respect to this publication or usage of information. Further, Sage reserves the right to make changes to the information described in this manual at any time without notice and without obligation to notify any person of such changes.
An Introduction to SData Endpoints
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Calls</td>
<td>15</td>
</tr>
<tr>
<td>Code Samples Using SData Client Libraries</td>
<td>16</td>
</tr>
<tr>
<td>System Adapter – Service Call</td>
<td>17</td>
</tr>
<tr>
<td>GetCurrentUser</td>
<td>17</td>
</tr>
<tr>
<td>Background</td>
<td>17</td>
</tr>
<tr>
<td>Sample Calls</td>
<td>17</td>
</tr>
<tr>
<td>Code Samples Using SData Client Libraries:</td>
<td>17</td>
</tr>
<tr>
<td>GCRM Endpoint</td>
<td>18</td>
</tr>
<tr>
<td>Purpose</td>
<td>18</td>
</tr>
<tr>
<td>Proxy Endpoints</td>
<td>18</td>
</tr>
<tr>
<td>Purpose</td>
<td>18</td>
</tr>
<tr>
<td>Background</td>
<td>18</td>
</tr>
<tr>
<td>Sample Calls</td>
<td>18</td>
</tr>
</tbody>
</table>
Introduction

Sage Data (SData) is the communication protocol that serves as a common language for interaction between Sage products worldwide. Sage products use SData to generate and consume feeds of information, similar to RSS feeds. For example, Financial Accounting/ERP applications can provide feeds containing sales orders, customer details, or project details. SData also allows Sage software to interact with the many third-party systems and services available through Web 2.0 technology.

- SData is a standards-based (HTTP, ATOM) specification.
- SData describes the mechanisms designed to support scenarios such as application integration, mobile clients, or mashups.

SData Endpoints

Windows Communication Foundation (WCF) defines an endpoint as a construct at which messages are sent or received (or both). It comprises a location (an address) that defines where messages can be sent, a specification of the communication mechanism (a binding) that described how messages should be sent, and a definition for a set of messages that can be sent or received (or both) at that location (a service contract).

SData builds Uniform Resource Identifiers (URIs), a compact string of characters used to identify or name a resource, to communicate with web services. These web services then query, insert, update, or delete information based on the URI sent.

- Supports GET, POST, PUT, and DELETE in both the Atom and Json/Bson formats. All values are editable.
- Supports the standard payload control options such as filtering, paging, and sorting.

There are four types of endpoints currently being offered through SData:

- Dynamic
- System
- GCRM
- Proxy
## Dynamic Endpoints

Dynamic endpoints provide create, read, update, and delete (CRUD) access to Sage SalesLogix entities. They can be used to query, filter, sort, and manipulate data through Sage SalesLogix or third-party applications. Dynamic endpoints are enabled by default when the SData portal is deployed. When SData is enabled for a Sage SalesLogix entity, code within the Application Architect generates a new dynamic endpoint.

### Call Parameters

**Where**
- Filters results

**select**
- Select specific items and excludes all others

**include**
- Include specific relationships in addition to others

**orderBy**
- Sort the results

### Endpoint Segments

SData uses URLs to address the resources, resource collections, schemas and operations that are exposed by a service.

http://www.example.com/sdata/slx/dynamic/prod/accounts?startIndex=21&count=10

<table>
<thead>
<tr>
<th>Example</th>
<th>Component Name</th>
<th>Description/Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>http</td>
<td>Protocol</td>
<td>https is also allowed.</td>
</tr>
<tr>
<td><a href="http://www.example.com">www.example.com</a></td>
<td>Server name</td>
<td>IP address is also allowed (192.168.1.1). Can be followed by port number. For example <a href="http://www.example.com:5493">www.example.com:5493</a>. 5493 is the recommended port number for SData services that are not exposed on the Internet.</td>
</tr>
<tr>
<td>sdata</td>
<td>Virtual Directory</td>
<td>Should be SData, unless the technical framework imposes something different.</td>
</tr>
<tr>
<td>six</td>
<td>Application</td>
<td>Name of the application.</td>
</tr>
<tr>
<td>dynamic</td>
<td>Contract name</td>
<td>An SData service can support several “integration contracts” side-by-side. For example, a typical Sage SalesLogix service will support system, dynamic, gcrm, and proxy contracts.</td>
</tr>
<tr>
<td>prod</td>
<td>Dataset</td>
<td>Identifies the dataset when the application gives access to several datasets, such as several companies and production/test datasets. If the application can only handle a single dataset, or if it can be configured with a default dataset, a hyphen can be used as a placeholder for the default dataset. For example, if prod is the default dataset in the example above, the URL could be shortened as: <a href="http://www.example.com/sdata/slx/dynamic/~accounts?startIndex=21&amp;count=10">http://www.example.com/sdata/slx/dynamic/~accounts?startIndex=21&amp;count=10</a>. If several parameters are required to specify the dataset (for example database name and company id), they should be formatted as a single segment in the URL. For example, slx/dynamic/demodb/acme/accounts -- the semicolon separator is application specific, not imposed by SData.</td>
</tr>
<tr>
<td>accounts</td>
<td>Resource Kind</td>
<td>This URL segment identifies the kind of resource that is queried (account, contact, salesOrder, etc.) This URL returns the collection of all account resources, as an Atom feed. If the contract exposes a large number of resources kinds, a functional group can be inserted before the resource kind segment (eventually a hierarchy of functional groups), which act as a folder (a hierarchy of folders) to organize the resources. Typical functional groups would be financials, commercials, HR, etc.</td>
</tr>
<tr>
<td>startIndex=21</td>
<td>Query parameters</td>
<td>The startIndex and count parameters allow the service consumer to request a specific page in a collection.</td>
</tr>
<tr>
<td>count=10</td>
<td>Query parameters</td>
<td></td>
</tr>
</tbody>
</table>

An Introduction to SData Endpoints
Sample Calls

List of dynamic feeds

http://servername:port/sdata/slx/dynamic/-/

Lookup Account by name

http://servername:port/sdata/slx/dynamic/-/Accounts?where=AccountName eq 'Above Marine'

Sort Contacts by last name

http://servername:port/sdata/slx/dynamic/-/Contacts?orderBy=LastName

Dynamic Endpoints in Sage SalesLogix v7.5.4

You can view a list of all of the enabled feeds by browsing to http://servername:port/sdata/slx/dynamic/-/. The list below shows the dynamic endpoints that have been enables in a non-customized v7.5.4 environment. Other entities can be enabled within the Application Architect. Instructions on how to enable SData feeds can be found in Generating SData Feeds for an Entity in the Application Architect Help files.

- accountProducts
- accounts
- accountSummaries
- activities
- addresses
- areaCategoryIssues
- attachments
- campaigns
- competitors
- contacts
- contracts
- countryCodeMappings
- deDupJobs
- deDupResults
- defectProblems
- defectReturns
- defects
- defectSolutions
- defectTickets
- events
- exchangeRate
- history
- leadAddresses
- leads
- leadSources
- opportunities
- opportunityContacts
- opportunityProducts
- ownerRights
- owners
- ownerSecurityProfiles
- packageProducts
- packages
- processes
- productPrograms
- products
- qualifications
- resources
- resourceSchedules
- returns
- roles
- salesOrderItems
- salesOrders
- sdxLogItems
- sdxPrices
- syncJobs
- ticketActivities
- ticketProblems
- tickets
- ticketSolutions
- unitOfMeasure
- urgencies
- userActivities
- userCalendars
- userInfo
- userNotifications
- userProfiles
- users

Creating Your Own Application

It is possible to use dynamic endpoints to create your own application using Visual Studio*. To build your own application, you will need to reference the Sage.SData.Client.dll in your application. You can copy this dll from the SData\bin folder in the virtual directory for the SData portal. The default location is C:\InetPub\wwwroot\Sdata\bin on the IIS Server where the SData portal is deployed. For more information on how to create your own application, please refer to the SData Master’s class.
System Endpoints

System endpoints, like dynamic endpoints, provide CRUD access to Sage SalesLogix data. Unlike Dynamic endpoints which are generated from code, System endpoints are generated from a fixed API.

Attachments system endpoint

Purpose
The Attachments system endpoint provides GET, POST, and PUT access to attachments in both Atom and Json/Bson formats. The endpoint supports standard payload options such as filtering, sorting, and paging.

Background
Attachments can be any documents, text files, or graphics that provide relevant information to an account, contact, opportunity, ticket, lead, contract, return, or defect. For more information, refer to the Attachments Tab topic in the Sage SalesLogix Client help system.

Endpoint Segments

<table>
<thead>
<tr>
<th>Segment</th>
<th>Name</th>
<th>Purpose</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachments</td>
<td>Resource kind</td>
<td>Returns all attachments</td>
<td></td>
</tr>
<tr>
<td>attachments('attachmentid')</td>
<td>Resource selector</td>
<td>Returns selected attachment</td>
<td>attachments('QDEMOA00036B')</td>
</tr>
<tr>
<td>attachments('attachmentid')/file</td>
<td></td>
<td>Downloads the file requested</td>
<td></td>
</tr>
<tr>
<td>Query parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>where</td>
<td></td>
<td>Filters feed results</td>
<td>attachments?where=fileName eq 'AbbottLtd2005gcr.pdf'</td>
</tr>
<tr>
<td>select</td>
<td></td>
<td>Select specific items and excludes all others.</td>
<td>/attachments?select=name</td>
</tr>
<tr>
<td>include</td>
<td></td>
<td>Include specific relationships in addition to others.</td>
<td></td>
</tr>
</tbody>
</table>

Errors and Warnings

If you request an attachment that does not exist, you get one of the following:
- an empty feed if a where condition is used
- an “entity does not exist” error diagnoses response if a predicate is used.

All the standard SData errors apply (for example, malformed URLs or payloads and missing IfMatch header during updates.)

Sample Calls

List all attachments:
/sdata/sl_ix/system/-/attachments

List all attachments ordered by file name:
/sdata/sl_ix/system/-/attachments?orderBy=filename

Lookup an attachment by ID:
/sdata/sl_ix/system/-/attachments('QDEMOA000646')
Lookup an attachment by file name:
/sdata/slx/system/-/attachments?where=fileName eq '2004FactBook.pdf'
Note that fileName shouldn't be used in predicate expressions since it isn't guaranteed to be unique.

Lookup all attachments for a specific account:
/sdata/slx/system/-/attachments?where=accountId eq 'AGHEA0002669'

Fetch an attachment file:
/sdata/slx/system/-/attachments('QDEMOA000033')?precedence=0
Precedence has been set to zero because we only want the associated file, not the entity payload.
If the server is configured correctly and the file can be found, then the response should be a multipart MIME message containing both the entity payload and the file contents.

Code Samples Using SData Client Libraries:
An attachment file can be downloaded, written to disk and opened as follows:

```csharp
var request = new SDataRequest("http://localhost/sdata/slx/system/-/attachments('QDEMOA000033')") {UserName = "admin"};
var response = request.GetResponse();
foreach (var file in response.Files)
{
    var filePath = Path.Combine(Path.GetTempPath(), file.FileName);
    using (var fileStream = File.OpenWrite(filePath))
    {
        var buffer = new byte[0x1000];
        int read;
        while ((read = file.Stream.Read(buffer, 0, buffer.Length)) > 0)
        {
            fileStream.Write(buffer, 0, read);
        }
    }
    Process.Start(filePath);
}
```

New attachments can be posted as follows:

```csharp
var filePath = @"C:\StringToColor.png";
var contentType = MimeHelper.FindMimeType(filePath);
var fileName = Path.GetFileName(filePath);
var stream = File.OpenRead(filePath);
var file = new AttachedFile(contentType, fileName, stream);
var operation = new RequestOperation(HttpMethod.Post, new AtomEntry()) {Files = {file}};
var request = new SDataRequest("http://localhost:2001/sdata/slx/system/-/attachments", operation) {UserName = "admin"};
request.GetResponse();
```
Groups System Endpoint

Purpose

The groups system endpoint provides GET, POST, PUT, and DELETE access to the groups attributes. It is a simple CRUD implementation not designed for complex group creation. The endpoint also supports standard payload options such as filtering, sorting, and paging. Group execution with a named query is supported using both GET and POST requests.

The groups system endpoint will return duplicates if the current user has access to multiple releases of a given group. When requesting a specific group’s metadata by family and name in predicate, duplicates will automatically be removed so that a single entry can be returned. The version and modify date properties are exposed so duplicates may be removed from the feed on the client-side.

Background

For general information on Groups, refer to the help topic Using Groups in the Sage SalesLogix Web Client help system. Sage SalesLogix has two kinds of groups: standard and ad hoc. Groups are created by individual users but can be shared with other users. Multiple versions and releases of a group may exist.

Endpoint Segments

<table>
<thead>
<tr>
<th>Segment</th>
<th>Name</th>
<th>Purpose</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>/groups</td>
<td>Resource</td>
<td>Returns all metadata on all groups for the logged in user</td>
<td>/groups</td>
</tr>
<tr>
<td></td>
<td>kind</td>
<td></td>
<td>/groups('groupid')</td>
</tr>
<tr>
<td>/groups('groupid')</td>
<td>Resource</td>
<td>Selects a specific group</td>
<td>/groups('p6UJ9A0004TS')</td>
</tr>
<tr>
<td></td>
<td>selector</td>
<td></td>
<td>Or any predicate expression, for example: /groups(name eq 'All Contacts')</td>
</tr>
<tr>
<td></td>
<td>Named</td>
<td>Executes the specified group</td>
<td>groups$queries/execute?_groupId=p6UJ9A0004TS</td>
</tr>
<tr>
<td></td>
<td>query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>where</td>
<td>filter</td>
<td>Filters feed results</td>
<td>/groups?where=mainTable eq 'Account'</td>
</tr>
<tr>
<td>select</td>
<td>select</td>
<td>Selects the properties that will be returned in the payload.</td>
<td>/groups?select=v</td>
</tr>
</tbody>
</table>

Errors and Warnings

If you request a group that does not exist, you get:

- an empty feed if a where condition is used
- an “entity does not exist” response if a predicate is used.

All the standard SData errors apply (for example, malformed URLs or payloads and missing IfMatch header during updates.)

Sample Calls

List available groups:

/slx/system/-/groups

Look up a specific group using its plugin ID:

/slx/system/-/groups('p6UJ9A0004TS')

Or any predicate expression, for example: /groups(name eq 'All Contacts')

Look up a specific group using its family and name:

/slx/system/-/groups(family eq 'Account' and name eq 'All Accounts')

Any predicate expression can be used so long as all the results have the same family and
name. If multiple releases or versions are found, then the latest is returned based on version and modifyDate.

List all groups for a given family:
/slx/system/-/groups?where=family eq 'Account'

List all groups ordered by their names:
/slx/system/-/groups?orderBy=name

List all groups that haven't been explicitly hidden by the current user:
/slx/system/-/groups?where=not isHidden

List all ad hoc groups:
/slx/system/-/groups?where=isAdHoc

List all ad hoc groups and include the entity IDs:
/slx/system/-/groups?where=isAdHoc&include=adHocIds

Executing Groups:
The payload is generated from the group layout. PERIOD, ISACTIVE and CONTRACTID are part of the payload for the Active Contracts group.

Execute a group based on its plugin ID:
/slx/system/-/groups/$queries/execute?_groupId=p6UJ9A0004TS
_groupId is a custom query argument. The SDATA specification requires that they be prefixed with an underscore.

Execute a group based on its family and name:
/slx/system/-/groups/$queries/execute?_family=Account&_name=All Accounts
Both _family and _name must be specified.

Execute a group with a custom order clause:
/slx/system/-/groups/$queries/execute?_groupId=p6UJ9A0004TS&orderBy=PERIOD desc

Execute a group with a custom filter clause:
/slx/system/-/groups/$queries/execute?_groupId=p6UJ9A0004TS&where=ISACTIVE eq 'T'

Execute a group with just the first column returned:
/slx/system/-/groups/$queries/execute?_groupId=p6UJ9A0004TS&select=CONTRACTID

Groups Database Information
The PLUGIN table contains the plugin information for groups created in Sage SalesLogix. Ad hoc groups are stored in the AdHocGroup table.
The PLUGIN table contains the plugin name, family, numeric type, user creating the plugin, date created or modified, version number of the plugin, company details, read only and other attributes.

Library System Endpoint

Purpose
The Library system endpoint provides GET, POST, PUT, and DELETE access to the Sage SalesLogix Library. The endpoint allows for filtering, sorting, and paging of the Library.
Background

The Library is a central repository for company information. It is used to provide the latest information to Sage SalesLogix users and to distribute information to Remote users. For more information, refer to the Library help topic in the Sage SalesLogix Administrator and Web Client help systems. The Library is managed in two tables, LIBRARYDIRS and LIBRARYDOCS. The LibraryDirs table stores the location (path) in relationship to the “Library” folder. (The path to the Library folder is stored in the BRANCHOPTIONS table.) The LibraryDocs table stores information about properties for each document or Web address URL that is in a Sage SalesLogix Library folder. Documents and URLs are added to the Library folder using the Administrator. The information includes file name, directory ID of the folder, description, available status, revision date, expiration date, file size, and a flag is set if the file can be sent to Remotes. The Abstract field can contain up to a 1024-character comment about the file.

Endpoint Segments

<table>
<thead>
<tr>
<th>Segment</th>
<th>Name</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>libraryDirectories</td>
<td>Resource kind</td>
<td>Returns all library directories</td>
<td></td>
</tr>
<tr>
<td>libraryDirectories(' microscopic world')</td>
<td>Resource selector</td>
<td>Returns selected directory</td>
<td>libraryDirectories('DDEMOA00000Y')</td>
</tr>
<tr>
<td>libraryDocuments</td>
<td>Resource kind</td>
<td>Returns all library documents</td>
<td></td>
</tr>
<tr>
<td>libraryDocuments(' microscopic world')</td>
<td>Resource selector</td>
<td>Returns selected document</td>
<td>libraryDocuments('dDEMOA000010')</td>
</tr>
</tbody>
</table>

Query parameters

- **where**
  - Filters feed results
  - libraryDirectories?where=fullPath eq "\Policies\ITravel" (note escaped backslashes)
  - /sdata/slix/system/-/libraryDirectories(fullPath eq "\Policies\ITravel")/documents?format=json

- **select**
  - Selects the properties that will be returned in the payload.

- **include**

Errors and Warnings

Requesting a library directory or document that does not exist, generates one of the following:

- an empty feed if a where condition is used
- an “entity does not exist” response if a predicate is used.

All the standard SData errors apply (for example, malformed URLs or payloads and missing IfMatch header during updates.)

Sample Calls

**List all documents:**

'/sdata/slix/system/-/libraryDocuments'

**List all directories:**

'/sdata/slix/system/-/libraryDirectories'
Get a directory by its full path:

/sdata/slx/system/-/libraryDirectories?where=fullPath eq '\Policies\Travel'

Notice that the backslashes are doubled up since backslash is the special escape sequence character.

Get a document including the directory it is in:

/sdata/slx/system/-/libraryDocuments('dDEMOA0000010')?include=directory

Get a directory including all the documents it contains:

/sdata/slx/system/-/libraryDirectories('DDEMOA00000Y')?include=documents

Code Samples Using SData Client Libraries:

The following code is for building a tree view with directory and document nodes and displaying the selected item in a property grid:

```csharp
var service = new SDataService("http://localhost/sdata/slx/system/-/", "admin", ");
var request = new SDataResourceCollectionRequest(service)
{
    ResourceKind = "libraryDirectories",
    QueryValues =
    {
        {"include", "documents"},
        {"orderBy", "directoryName"}
    }
};
var treeView = new TreeView {Dock = DockStyle.Fill};
var propGrid = new PropertyGrid {Dock = DockStyle.Fill};
var expand = new Action<string, TreeNodeCollection>((id, nodes) =>
{
    request.QueryValues["where"] = string.Format("parentId eq '{0}'", id);
    request.ServerName = "";
    request.ServerName = "localhost";
    foreach (var entry in request.Read().Entries)
    {
        var dir = entry.GetDataPayload();
        var node = new TreeNode((string)dir.Values["directoryName"])
        {
            Tag = dir,
            Nodes = {new TreeNode()}
        };
        nodes.Add(node);
    }
    expand("0", treeView.Nodes);
    treeView.BeforeExpand += (sender, e) =>
```
{ if (e.Node.Nodes.Count > 0 && e.Node.Nodes[0].Tag == null) {
    var dir = (SDataPayload) e.Node.Tag;
    e.Node.Nodes.Clear();
    expand(dir.Key, e.Node.Nodes);
    foreach (var file in (SDataPayloadCollection) dir.Values["documents"])
    {
        e.Node.Nodes.Add(new TreeNode((string) file.Values["fileName"])
        {
            Tag = file,
            ForeColor = Color.Blue
        });
    }
}
}
treeView.AfterSelect += (sender, e) => propGrid.SelectedObject = e.Node.Tag;
new Form
{
    Controls =
    {
        new SplitContainer
        {
            Dock = DockStyle.Fill,
            Panel1 = {Controls = {treeView}},
            Panel2 = {Controls = {propGrid}}
        }
    }
}.ShowDialog();

PickLists System Endpoint

Purpose
The pick list system endpoint provides GET, POST, PUT, and DELETE access to the pick list attributes in both Atom and Json/Bson formats. The endpoint also supports standard payload options such as filtering, sorting, and paging. A relationship property endpoint is available for pick list items. Access to pick list items is also made possible through the use of nested payloads.

Background
A pick list definition consists of attribute settings and list items. User interfaces apply pick list attribute settings to enforce security and business rules for individual pick lists. For example, there are attributes that specify whether the items of a pick list are editable and whether multiple items can be selected. Pick lists are administered through the Pick List Manager which is available to the system administrator and users who have been assigned the appropriate administrative role. Pick lists can also be administered in the Sage SalesLogix Web Client.
Endpoint Segments

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>/pickLists</td>
<td>Resource kind</td>
<td>Returns attribute information for all picklists</td>
<td>/pickLists('kSYST0000331') or /pickLists(id eq 'kSYST0000383')</td>
</tr>
<tr>
<td>('picklistid')</td>
<td>Resource selector</td>
<td>Selects a specific pick list</td>
<td>/pickLists('kSYST0000331')/items</td>
</tr>
<tr>
<td>/items</td>
<td>Relationship property endpoint</td>
<td>Returns items of the selected pick list without the pick list attribute information</td>
<td>/pickLists('kSYST0000331')/items</td>
</tr>
</tbody>
</table>

Query parameters

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>where</td>
<td>Filters feed results</td>
<td></td>
<td>/pickLists?where=name like '%Skill%'</td>
</tr>
<tr>
<td>select</td>
<td>Selects the properties that will be returned in the payload.</td>
<td></td>
<td>/pickLists?select=name</td>
</tr>
<tr>
<td>include</td>
<td>Expands children in the payload.</td>
<td></td>
<td>/pickLists('kSYST0000331')?include=items</td>
</tr>
</tbody>
</table>

Errors and Warnings

Requesting a pick list that does not exist, generates one of the following:
- an empty feed if a where condition is used
- an “entity does not exist” response if a predicate is used.

All the standard SData errors apply (for example, malformed URLs or payloads and missing IfMatch header during updates.)

Sample Calls

**List all pick lists:**

/slx/system/-/pickLists

**Lookup a pick list by ID:**

/slx/system/-/pickLists('kSYST0000385')

Or any predicate expression that returns a single resource, for example:

/pickLists(name eq 'Skill Category')

**Lookup a pick list by name:**

/slx/system/-/pickLists(name eq 'Region')

**Include the pick list items in the payload with the pick list attribute values:**

/slx/system/-/pickLists?include=items

**List all items for a pick list by ID (without the pick list attributes):**

/slx/system/-/pickLists('kSYST0000316')/items

**List all items for a pick list by name:**

/slx/system/-/pickLists(name eq 'Opportunity Cycle')/items
Select only the text for each item:

/slx/system/-/pickLists('kSYST0000316')/items?select=text

Order items by the definition sort order:

/slx/system/-/pickLists('kSYST0000316')/items?orderBy=sort

Order items alphabetically:

/slx/system/-/pickLists('kSYST0000316')/items?orderBy=text

Code Samples Using SData Client Libraries:

```javascript
var service = new SDataService("http://localhost/sdata/slx/system/-/") {UserName = "admin"};
var request = new SDataSingleResourceRequest(service) {
    ResourceKind = "pickLists",
    Include = "items"
};

//CREATE
var entry = new AtomEntry();
var items = new SDataPayloadCollection {
    new SDataPayload {
        ResourceName = "pickListItem",
        Values = {
            {"text", "_Text1"},
            {"code", "_Code1"},
            {"number", "1"}
        }
    },
    new SDataPayload {
        ResourceName = "pickListItem",
        Values = {
            {"text", "_Text2"},
            {"code", "_Code2"},
            {"number", "2"}
        }
    }
};
var payload = new SDataPayload {
    ResourceName = "picklist",
    Values = {
        {"name", "_Name"},
        {"allowMultiples", "true"},
        {"valueMustExist", "true"},
        {"required", "true"},
        {"alphaSorted", "true"},
        {"noneEditable", "true"},
        {"items", items}
    }
};
```
entry.SetSDataPayload(payload);
request.Entry = entry;
entry = request.Create();

//UPDATE
payload = entry.GetSDataPayload();
items = (SDataPayloadCollection) payload.Values["items"];  
request.Entry = entry;
request.ResourceSelector = string.Format("'{0}'", payload.Key);
payload.Values["allowMultiples"] = "false";
payload.Values["valueMustExist"] = "false";
payload.Values["required"] = "false";
payload.Values["alphaSorted"] = "false";
payload.Values["noneEditable"] = "false";
items[0].Values["code"] += "_CHANGED";
items[0].Values["number"] += "0";
items[1].Values["code"] += "_CHANGED";
items[1].Values["number"] += "0";
entry = request.Update();

//DELETE
request.Entry = entry;
request.Delete();

**SystemOptions System Endpoint**

**Purpose**
The systemOptions system endpoint provides Read-Only access to system level information such as whether multi-currency or Unicode is enabled. This endpoint returns one entry with a property for each option exposed.

**Background**
System level information such as Company Name and database version is stored in the SYSTEMINFO table in the Sage SalesLogix database. It also contains a blob field that holds information such as default password, database admin id, and current remote password. The systemOptions system endpoint returns information from this table. For security, the systemOptions endpoint is Read-Only.

**Errors and Warnings**
All the standard SData errors apply (for example, malformed URLs or payloads and missing IfMatch header during updates.)

**Sample Calls**

**List all information**
/sdata/slx/system/-/systemOptions

**List select information**
/sdata/slx/system/-/systemOptions?select=,name,value
Get company name
/sdata/slx/system/-/systemOptions('CompanyName')

Get version of database
/sdata/slx/system/-/systemOptions('DatabaseVersion')

Get company name and version of database
/sdata/slx/system/-/systemOptions?where=name eq 'CompanyName' or name eq 'DatabaseVersion'

Code Samples Using SData Client Libraries

```csharp
public string GetGlobalOption(string optionName)
{
    string resp = "";
    ISDataService service = NewSDataService("system");
    var request = new SDataResourcePropertyRequest(service);
    request.ResourceKind = "systemOptions";
    request.ResourceSelector = "'" + optionName + "'";
    try
    {
        AtomEntry entry = request.Read();
        SDataPayload payload = entry.GetSDataPayload();
        if (payload.Values.Count > 0)
        {
            resp = (string)payload.Values["value"]; // Assuming the value is a string
            if (resp == null) resp = "";
        }
    }
    catch (Exception)
    {
        resp = "";
    }
    return resp;
}
```

UserOptions System Endpoint

Purpose
The userOptions system endpoint provides GET, POST, PUT, and DELETE access to the userOptions attributes in both Atom and Json/Bson formats. The endpoint also supports standard payload options such as filtering, sorting, and paging.

The userOptions system endpoint uses the special composite key format to request items. Short and normal key formats are provided for queries. For example, `/userOptions('General;Currency')` is equivalent to `/userOptions('Category=General;Name=Currency')`. For security reasons, users will only be able access their own user options through this endpoint.

Background
Sage SalesLogix provides tools to allow users to modify some parts of the interface so that they can work in the way that suits them best. The Sage SalesLogix Web Client User Options allow users to set specific preferences for such features as the view that appears when they start the Sage SalesLogix
Web Client and their Activity Reminders. The settings that a user enters are recorded as being for the user who is logged on and are not visible to other Sage SalesLogix users.

Sage SalesLogix code defines and stores multiple user options. This endpoint enables you to get a list of the user options available. For more information on user options, refer to the help topic, Editing User Options in the Web Client help system.

As you create your own customizations, you may identify additional user preferences that you want to persist. For example, you may want to remember the default tab for a new form or when a user turns off future display of a particular message box. You can add custom user options to track this information.

User options are managed in two tables, USEROPTIONDEF and USEROPTIONS. The USEROPTIONDEF table stores options, definitions, and default values. The name of the option in combination with the option category provides the primary key and must be unique. The USEROPTIONS table stores values that override the default values for individual users. In order for a value to exist in userOptions, a corresponding definition of the option should exist in the USEROPTIONDEF table.

The USEROPTIONDEF table has a composite key based on category and name.

- Queries are based on the USEROPTIONDEF table so “dangling” user options (which only exist in USEROPTIONS) are not exposed.

You can also see which user options are stored by browsing the Sage SalesLogix-defined options in the Sage SalesLogix database.

### Endpoint Segments

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>/userOptions</td>
<td>Resource kind</td>
<td>Returns all user options for logged in user</td>
<td>/userOptions</td>
</tr>
<tr>
<td>(&quot;Category;Name&quot;)</td>
<td>Resource selector</td>
<td>Selects a specific user option for logged in user</td>
<td>/userOptions(&quot;General;Currency&quot;) /userOptions(&quot;category=General;name=Currency&quot;)</td>
</tr>
<tr>
<td>Query parameters</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>where</td>
<td>Filters feed results</td>
<td>/userOptions?where=category eq 'General'</td>
<td></td>
</tr>
<tr>
<td>select</td>
<td>Selects the properties that will be returned in the payload. Properties are category, name, displayName, defaultValue, value, and locked.</td>
<td>/userOptions?select=value</td>
<td></td>
</tr>
</tbody>
</table>
List all options for a specific category
/slx/system/-/userOptions?where=category eq 'General'

Get a specific option using the short key format
/slx/system/-/userOptions('General;Currency')
Or any predicate expression, for example:
/userOptions(category eq 'General' and name eq 'Currency')

Get a specific option using the normal key format
/slx/system/-/userOptions('category=General;name=Currency')

Get a specific option using an expression
/slx/system/-/userOptions(category eq 'General' and name eq 'Currency')

Get just the value of a specific user option
/slx/system/-/userOptions('category=General;name=Currency')?select=value

Code Samples Using SData Client Libraries

```csharp
var service = new SDataService("http://localhost/sdata/slx/system/-/") { UserName = "admin"};
var request = new SDataSingleResourceRequest(service) { ResourceKind = "userOptions"};

//CREATE
var entry = new AtomEntry();
var payload = new SDataPayload
{
    ResourceName = "userOption",
    Values =
    {
        {"category", "_Category"},
        {"name", "_Name"},
        {"displayName", "_DisplayName"},
        {"defaultValue", "_DefaultValue"},
        {"value", "_Value"},
        {"locked", "true"}
    }
};
entry.SetSDataPayload(payload);
request.Entry = entry;
entry = request.Create();

//UPDATE
payload = entry.GetSDataPayload();
request.Entry = entry;
request.ResourceSelector = string.Format("'{0}'", payload.Key);
payload.Values["displayName"] += "_CHANGED";
payload.Values["defaultValue"] += "_CHANGED";
payload.Values["value"] += "_CHANGED";
```
System Adapter – Service Call

SData supports service operations. Service operations are operations that do not fit naturally into the CRUD model. For more information about service calls, see http://interop.sage.com/daisy/sdata/ServiceOperations.html

GetCurrentUser

Background

The getCurrentUser service operation is a service that returns the userId, userName, and prettyName properties of the application’s current logged in user from the IUserService. In cases where the Sage SalesLogix user is different than the login credentials, the getCurrentUser service operation can be used to retrieve the current user id needed by service calls to post the correct data.

Sample Calls

GET template:
/sdata/slx/system/-/$service/getCurrentUser/$template

POST
/sdata/slx/system/-/$service/getCurrentUser

Code Samples Using SData Client Libraries:

```javascript
var service = new SDataService("http://localhost/sdata/slx/system/-/");
var request = new SDataServiceOperationRequest(service)
{
    OperationName = "getCurrentUser",
    Entry = new AtomEntry()
};
var entry = request.Create();
var payload = entry.GetSDataPayload();
payload = (SDataPayload) payload.Values["response"];
MessageBox.Show("UserID: " + payload.Values["userId"]
    + "\nUser Name: " + payload.Values["userName"]
    + "\nName: " + payload.Values["prettyName"], "User Information");
```
GCRM Endpoint

Purpose
The Global CRM (GCRM) contract is a common interface between CRM products and ERP products in Sage. The aim of the contract is to enable ERP teams to develop one integration solution which works with Sage ACT!, Sage SalesLogix, and Sage CRM. The contract defines a common set of functionality that is specifically designed to ease the development and running of point-to-point integrations with ERP products.

Provided the specification is adhered to it is possible to synchronize trading accounts, customer, supplier, sales orders, sales quotations, commodity and other ERP information with CRM to a reasonably sophisticated level without significant additional customization or coding.

Sage SalesLogix provides GCRM endpoints so that it can synchronize information to an ERP system. This will allow for fast, accurate updates between Sage SalesLogix and your accounting program. You can view a list of the provided endpoints by browsing to http://servername:port/sdata/slx/gcrm/-. For more information on the GCRM contract and how you can take advantage of it, please see http://interop.sage.com/daisy/SGCRMContract/274-DSY.html.

Proxy Endpoints

Purpose
Proxy endpoints provide access to SData feeds for external applications. Information the proxy endpoint uses to access the external feeds is stored in the AppIdMapping table. For more information on how to add external web service information, see “Using an external feed with an Editable Grid Control – Example” in the Sage SalesLogix Application Architect Help system.

Background
The AppIdMapping table in the Sage SalesLogix database contains the information that is needed for the proxy endpoint to access external SData feeds. Adding the information for external feeds can be done by logging into the Sage SalesLogix Web Client as admin. The table contains the appId, name, endpointUrl, username, and password for accessing the feed.

Sample Calls

Feed containing product details:
/sdata/slx/proxy/(appid)/products

Feed containing customer details:
/sdata/slx/proxy/(appid)/customers