



## Getting to Know InfoTables

---

The InfoTable is an often used and sometimes misunderstood data structure in the Thingworx platform. Many Service calls use InfoTables as input parameters and output return values so there is no avoiding them and once you get hang of them, you may even want to use them in your own Things.

### Audience

A new ThingWorx developer who has had some exposure to the Composer, modeling and the writing of Services and wants to learn more about InfoTables. This document discusses features available in ThingWorx version 5.2 or higher.

### Contents

What is an InfoTable? .....	2
So how do Arrays work In JavaScript? .....	2
How is an InfoTable different than an Array? .....	3
What is a DataShape? .....	3
How do you create an InfoTable? .....	3
What is an InfoTable (Again).....	6
Playing by the InfoTable Rules .....	6
Two ways to add an Object to an InfoTable .....	7
Dynamically creating or modifying the DataShape of an InfoTable .....	8
What is the best way to manipulate an InfoTable? .....	9
How do you remove all the objects from an InfoTable? .....	10
Creating InfoTables directly from DataShapes .....	10
What can I do with InfoTables? .....	11
Some Examples .....	11
Adding an object based on a DataShape to a Stream.....	11
Displaying an InfoTable in a Mashup using a Grid Widget .....	12
Declaring a Thing Property of Base Type InfoTable .....	15
Operations that can be performed on InfoTables.....	17
Conclusion.....	21

## What is an InfoTable?

**An InfoTable is zero indexed, ordered array of JavaScript objects which expose *same* properties.**

Since an InfoTable is an Array of Objects, it can also be thought of as a table where the array entries are rows and the object properties of each object in the array are the columns. Since we are trying to represent tabular data it is important that each object have the same properties so that each column of the table can map to a property value.

## So how do Arrays work In JavaScript?

If you have ever worked with an Array object in JavaScript you are familiar with how arrays work. For those who may not be lets go over the basics.

Declaring an array

```
var myArray=new Array();  
var mySecondArray=[ ];
```

Both of the above statements are equivalent. They both create array objects.

An array can contain multiple objects in sequence of any type. For example

```
var myArrayOfStuff=["Pickel",3,{name:"Bill",phone:"555-1212"}];
```

The above array contains a String, a number and a custom object with the fields name and phone. If you want to try this out for yourself using the Chrome browser run it and open the Javascript Console (control-shift-J) and paste in the line above to create this array and try these examples to access its items. Below is a screen shot of the JavaScript console in Chrome showing how to create and then access each of the objects in the JavaScript array.

```
> var myArrayOfStuff=["Pickel",3,{name:"Bill",phone:"555-1212"}];  
< undefined  
> myArrayOfStuff[0]  
< "Pickel"  
> myArrayOfStuff[1]  
< 3  
> myArrayOfStuff[2]  
< Object {name: "Bill", phone: "555-1212"}  
> myArrayOfStuff[2].name  
< "Bill"
```

The “undefined” value is not an error on line two above. The declaration of a local scope variable (var) returned the object undefined as a result but it does create the variable.

### How is an InfoTable different than an Array?

In the example above, the JavaScript array contains completely different types of objects. InfoTables can only contain the same type of object. This type is defined, not with JavaScript’s native object typing mechanism but through the use of another ThingWorx specific typing definition called a DataShape. JavaScript considers every type of object you store in an InfoTable to be of type “Object” as you can see above when the command myArrayOfStuff[2] is printed out. InfoTables require that each object in the array expose the exact same properties.

### What is a DataShape?

**A DataShape is a specification of the required property names and return types each property of a JavaScript “Object” must have to be added to an InfoTable.**

DataShapes also contain other metadata about a property that is useful for formatting and displaying the InfoTable in a tabular format or Grid. DataShapes can be declared in your ThingWorx composer and are required to build a functional InfoTable. They are essentially a schema mechanism for defining the required fields of a Thing in ThingWorx.

### How do you create an InfoTable?

We talked about creating JavaScript arrays but how do you create an InfoTable and how is it different than a JavaScript array? Well it starts with a DataShape. Lets create a simple one. Since you are already familiar with ThingWorx, below is a screen shot of a simple DataShape defining three fields, name:string, phone:string and age:number. Go ahead and create this DataShape for yourself.

The screenshot shows the SimpleDataShape web interface. At the top, there is a logo with three colored spheres (yellow, blue, red) and the text 'SimpleDataShape'. To the right of the logo is a 'DataShape' label with a question mark icon, and an orange 'Edit' button. Below the top bar is a sidebar on the left with three main sections: 'ENTITY INFORMATION', 'PERMISSIONS', and 'Fields'. Under 'ENTITY INFORMATION', there are three sub-items: 'General Information' (with an info icon), 'Field Definitions' (which is highlighted in blue), and 'PERMISSIONS'. Under 'PERMISSIONS', there are three sub-items: 'Visibility' (with a globe icon), 'Design Time' (with a clock icon), and 'Run Time' (with a document icon). The 'Fields' section is currently active, showing a table with two columns: 'Name' and 'Additional Info'. The table contains three rows: the first row has 'name' in the 'Name' column and an empty cell in the 'Additional Info' column; the second row has 'phone' in the 'Name' column and an empty cell; the third row has 'age' in the 'Name' column and an empty cell. Each row has a small icon to the left of the name: a minus sign for 'name', a minus sign for 'phone', and a hash symbol for 'age'.

Now that you have defined this DataShape, you can declare an InfoTable which can contain multiple objects that conform to this DataShape.

**Note:** *It is common for ThingWorx to wrap all its Objects inside an InfoTable even if there is only one object in the InfoTable because knowing the DataShape of that single object makes it easy to display in the composer.*

Create any service and use the “Create InfoTable from selected data shape” snippet and you will see this code generated.

```
var params = {
    infoTableName : "InfoTable",
    dataShapeName : "SimpleDataShape"
};

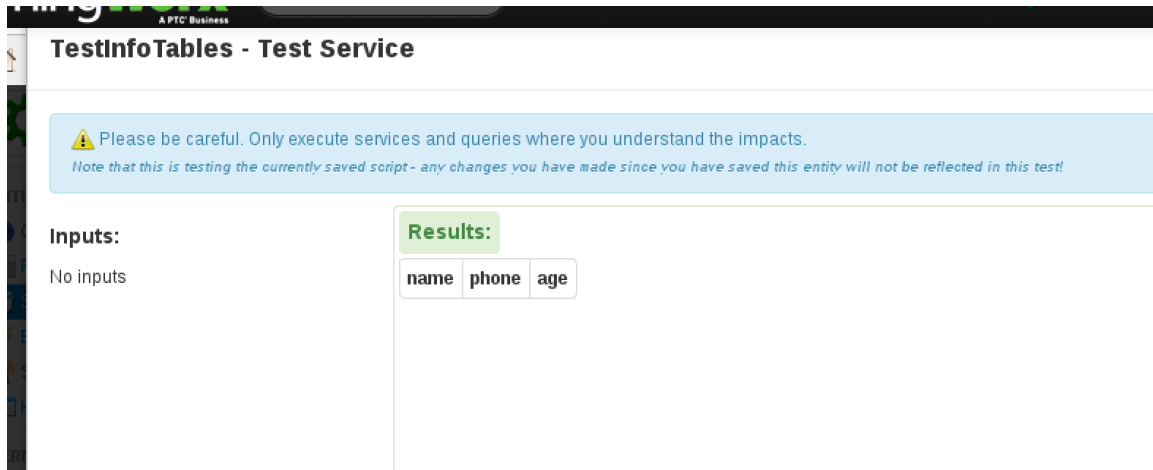
var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);
```

Which can also be restated in a more compact form as:

```
var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape({
    infoTableName : "InfoTable",
    dataShapeName : "SimpleDataShape"
});
```

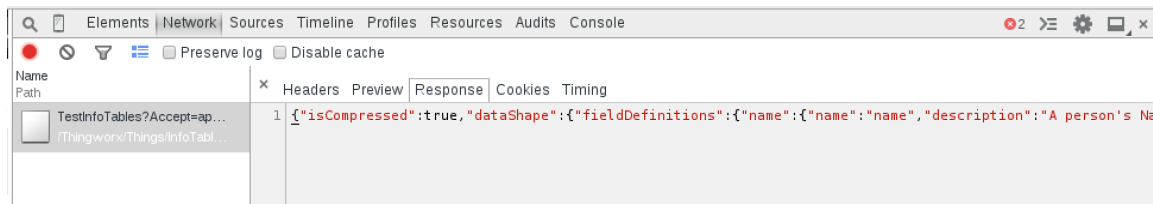
Services always treat the result local scope variable as their return value so when you actually test run this service, the resulting InfoTable will be returned as its output. Make sure you declared the output type of the Service as InfoTable.

The results, shown below will not amaze you.



You get an empty table with columns named after the DataShape and no data but there is more going on here than you are seeing. You have created an InfoTable and using the Chrome developer tools, you can finally see what an InfoTable consists of.

In Chrome, open the Developer Tools (ctrl-shift-I) and select the Network tab. Keep this tool open and retest this service. Now look at the new row representing your test in the network tool.



You can see that there actually was a complex JavaScript object returned by this service that created the bland, empty InfoTable you saw in HTML. The returned object is listed below.

```
{
  "isCompressed":true,
  "dataShape":{
    "fieldDefinitions":
    {"name":{"name":"name","description":"A person's
Name","baseType":"STRING","ordinal":1,"alias":"_0","aspects
":{}},
    "phone":{"name":"phone","description":"A Phone
number","baseType":"STRING","ordinal":2,"alias":"_1","aspec
ts":{}},
    "age":{"name":"age","description":"","baseType":"NUMBE
R","ordinal":3,"alias":"_2","aspects":{"maximumValue":100.0
,"minimumValue":1.0}}}},
    "rows":[]
  }
}
```

Now this is much more revealing. We can now describe an InfoTable in clearly defined terms.

## What is an InfoTable (Again)

The object declaration above is often referred to as JSON format. JSON stands for JavaScript Object Notation and is also a valid way to declare objects in JavaScript. By looking at the JSON above you can see that an InfoTable consists of a JavaScript Object with three properties. The important properties are **.dataShape** and **.rows**. From the discussion above you can see that **.rows** is a JavaScript Array with zero objects in it.

Now take a look at the JSON format of a DataShape. A DataShape is a JavaScript Object with a property called **.fieldDefinitions** which has one property named for each required property that must be present in any object added to the **.rows** array. Each property contains an object that provides descriptive data about the data type that this field must contain. The DataShape, in general, is used to describe the kind of Objects that the **.rows** Array can contain.

If you look at the snippet, “Create Empty Info Table” now, the returned code should make perfect sense.

```
var result = { dataShape: { fieldDefinitions : {} }, rows:
[] };
```

This InfoTable is not very useful since its DataShape has not been assigned and it won't support any helper functions like **AddRow()** or **getRow()** since those functions don't exist on this object. This object is just data, no functions.

## Playing by the InfoTable Rules

Lets create an InfoTable with a valid DataShape in a Service again and this time, populate it with data using snippets.

```
// Snippet 1.Create Info Table from selected Data Shape
var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape({
    infoTableName : "InfoTable",
    dataShapeName : "SimpleDataShape"
});
```

Now the results object has a populated **result.dataShape** and an empty **result.rows** properties so this InfoTable is ready to have objects added to its **.rows** array. As we saw above, creating a DataShape object is not trivial and the API call **CreateInfoTableFromDataShape** has just done all the work for us. Now lets add an item to the **.rows** array that conforms to the **SimpleDataShape** using a snippet.

```
// Snippet 2. Create InfoTable Entry from Datashape
var newEntry = new Object();
newEntry.phone = undefined; // STRING
newEntry.age = undefined; // NUMBER
newEntry.name = undefined; // STRING
```

Replace the undefined objects that the snippet generates with real values.

```
// Snippet 2. Create InfoTable Entry from Datashape
var newEntry = new Object();
newEntry.phone = "555-555-5555"; // STRING
newEntry.age = "22"; // NUMBER
newEntry.name = "John Smith"; // STRING
```

or an alternative way to write this would be to just declare the object using JSON style notation.

```
var newEntry = {
  phone: "555-555-5555",
  age: "22",
  name: "John Smith"
};
```

Now that you have created a JavaScript object that conforms to the DataShape, you can add it to the **.rows** Array one of two ways. You could add the row by pushing it onto the **.rows** array or you could use one of the InfoTable API Services in the Resources Thing.

## Two ways to add an Object to an InfoTable

Lets resume the discussion of populating an InfoTable. Below, we will create an info table now without snippets, using the most compact notation possible and add one object to it.

```
// Using the InfoTable API call, Addrow()
var myInfoTable =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape({
  infoTableName : "InfoTable",
  dataShapeName : "SimpleDataShape"
});

myInfoTable.AddRow({
  phone: "555-555-5555",
  age: "22",
  name: "John Smith"
});
var result=myInfoTable; // So the service returns a value
```

.AddRow() will only be available if your InfoTable was created using Resources["InfoTableFunctions"].CreateInfoTableFromDataShape() because the returned JavaScript Object has the .AddRow() function added to the returned InfoTable. Here is an alternative way to make an InfoTable without using any API calls. It also would allow you to change the DataShape of an info table on the fly.

```
var myInfoTable = {
    rows:[],
    dataShape:
        DataShapes["SimpleDataShape"].GetDataShapeMetadataAsJSON()
};

myInfoTable.rows.push({
    phone:"555-555-5555",
    age:"22",
    name:"John Smith"
});

var result=myInfoTable;
```

When you run this service, it produces a valid InfoTable but does not have the .AddRow() function since the InfoTable was not created by the API call Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(). The push function comes from the JavaScript Array object and adds a new row to the Array.

### Dynamically creating or modifying the DataShape of an InfoTable

In a previous section, we demonstrated that you can create an empty InfoTable with the code below.

```
var myInfoTable = { dataShape: { fieldDefinitions : {} } ,
rows: [] };
```

Here there will be no helper functions such as .AddRow() available for you to use but this is still a valid InfoTable to use anywhere in ThingWorx. Lets add an object to this InfoTable and modify this empty DataShape on the fly to support this object.

```
var budget1= { department:"marketing", budget:2500 };
```

Now we will need to add the “department” and “budget” properties to this DataShape before we can add the budget1 object to this InfoTable. Below is the entire example as a service. Make sure when you try it to set the output to INFOTABLE.

```
var myInfoTable = { dataShape: { fieldDefinitions : {} } ,
rows: [] };
var budget1= { department:"marketing", budget:2500 };
```



```

myInfoTable.dataShape.fieldDefinitions["department"] =
{
    name: "department",
    baseType: "STRING"
};
myInfoTable.dataShape.fieldDefinitions["budget"] =
{
    name: "budget",
    baseType: "NUMBER"
};

// Now you can add budget1 to the InfoTable.
myInfoTable.rows.push(budget1);

// Return this InfoTable as the output
var result=myInfoTable;

```

Other values for "baseType" can be found in the Composer in the properties panel. After seeing this example you can now see the potential for adding or removing properties from DataShape at any time.

This same process can be done with the DataShape API on persisted DataShapesthat you create in the Composer. The example below would modify an existing DataShape called TestDataShape giving all the objects it describes two new properties.

```

DataShapes["TestDataShape"].AddFieldDefinition(
    {
        name: "department" ,
        dataShape: "department" ,
        type: "STRING"
    }
);
DataShapes["TestDataShape"].AddFieldDefinition(
    {
        name: "budget" ,
        dataShape: "budget",
        type: "NUMBER"
    }
);

```

### What is the best way to manipulate an InfoTable?

Using the InfoTableFunctions is the preferred way to construct InfoTables as the InfoTables they create will have helper functions that avoid you ever having to access the **.rows** property directly. Below is a table that demonstrates these helper functions versus manipulating the .rows array directly

API Created InfoTable Operations	Manually Created InfoTable Operations
<code>myInfoTable.AddRow({   phone:"555-555-5555",   age:"22",   name:"John Smith" })</code>	<code>myInfoTable.rows.push({   phone:"555-555-5555",   age:"22",   name:"John Smith" })</code>
<code>myInfoTable.getRow(0)</code>	<code>myInfoTable.rows[0]</code>
<code>myInfoTable.RemoveRow(rowIndex);</code>	<code>myInfoTable.rows.splice(rowIndex,   1);</code>
<code>myInfoTable.getRowCount()</code>	<code>myInfoTable.rows.length</code>

In each case above, using an API constructed InfoTable prevents you from having to access the `.rows` property or even being aware it exists. Discussing this alternative method is really only useful as a learning exercise.

### How do you remove all the objects from an InfoTable?

This may seem too simple for its own section but the last thing you should try to do is call `myInfoTable.RemoveRow()` from a for loop to remove all its items. Never remove objects from an array while iterating it like this because its size is shrinking on each pass of the loop and you will get an index out of bounds exception.

```
// !!! Never do this !!!
for(var index=0;index< myInfoTable.getRowCount();index++){
  myInfoTable.RemoveRow(index);
}
```

Do something like this

```
while(myInfoTable.getRowCount(>0){
  myInfoTable.RemoveRow(0);
}
```

or even this if you don't have any other references directly to the rows array.

```
myInfoTable.rows=[];
```

### Creating InfoTables directly from DataShapes

It turns out there is a handy function called `CreateValues()` that is present on the DataShapes you define that will produce an InfoTable to contain objects with that shape. Here are some examples.

```
// Use the global DataShapes collection to find your
// SimpleDataShape thing and call .CreateValues() on it.

var result=DataShapes["SimpleDataShape"].CreateValues();
```

Result now contains an empty InfoTable ready to receive objects with the DataShape "SimpleDataShape".

But why stop there? Lets use another version of this function to both create and populate your InfoTable with an initial object using CreateValuesWithData().

```
var result = DataShapes["SimpleDataShape"].CreateValuesWithData(  
    {  
        values: {  
            phone: "666-666-6666",  
            age: "35",  
            name: "Will Smith"  
        }  
    }  
);
```

Result now contains a valid InfoTable with one row in it, constructed from a DataShape.

### What can I do with InfoTables?

You can do many useful things with InfoTables. Here are some examples.

- Some ThingWorx services require an InfoTable with one item in them as their only input parameter.
- InfoTables can be displayed in the Grid Widget in a mashup
- InfoTables are a Thing property base type. That means that you can declare properties that are collections of a DataShape types instead of just using Strings or Numbers. If you configure them as Persistent then the contents of the InfoTable will be saved permanently.
- InfoTables created within a service call can be manipulated, searched and filtered allowing your service to perform custom operations on InfoTable data.
- InfoTables can be converted into XML documents for the creation of Web Services.

### Some Examples

#### Adding an object based on a DataShape to a Stream.

A Stream is a sequential persisted collection of objects of a particular DataShape with a timestamp, location and collection of tags for each added object. Create a Stream called SimpleDataStream which uses SimpleDataShape as its DataShape with the composer. Now the following code will log an object to this stream.

```
var params = {  
    tags : [],
```

```

timestamp : new Date(), // The time now
source : me.name, // The name of thing that creates this entry
values : DataShapes["SimpleDataShape"].CreateValuesWithData(
  {
    values: {
      phone:"666-666-6666",
      age:"35",
      name:"Will Smith"
    }
  }
),
location : {latitude:0,longitude:0,elevation:0,units:"WGS84"}
};

Things["SimpleDataStream"].AddStreamEntry(params);

```

Now that you know more about InfoTables, the example above should become clear. Here, this service requires the parameter, values, which is in InfoTable containing objects that are based on the DataShape the stream expects to store. We use the DataShape to create this InfoTable.

The Stream itself also implements a service called CreateValuesWithData() which does the same thing so Things["SimpleDataStream"].CreateValuesWithData() would have worked just as well to construct the InfoTable.

### Displaying an InfoTable in a Mashup using a Grid Widget

By now you should be getting quite comfortable creating InfoTables. Lets create a Service that returns an InfoTable containing three objects based on the SimpleDataShape we have been using all along.

Create a new Thing based on GenericThing called InfoTableTestThing. Create a service on this thing called GetPeople.

Have it output an InfoTable based on SimpleDataShape.

**New Service** ? Local (JavaScript) ▾

Service Info **Inputs/Outputs** Snippets Me Entities

**Inputs** + Add

Name	Actions
No Inputs	

**Outputs** ?

Name ? result

Description ?

Base Type ? INFOTABLE ▾

Data Shape: ? SimpleDataShape ✕

Now the service Javascript should look like this.

```
var myInfoTable= DataShapes["SimpleDataShape"].CreateValues();
myInfoTable.AddRow(
    {phone:"666-666-6666",
      age:"35",
      name:"Will Smith"
    });
myInfoTable.AddRow(
    {phone:"555-555-1212",
      age:"21",
      name:"Bill Smith"
    });
myInfoTable.AddRow(
    {phone:"222-555-5555",
      age:"12",
      name:"Phil Smith"
    });
var result=myInfoTable;
```

When you test this service, you will get a formatted table as a result.

## GetPeople - Test Service

**⚠ Please be careful. Only execute services and queries where you understand the impacts.**  
*Note that this is testing the currently saved script - any changes you have made since you have saved this entity will not be reflected in this test!*

### Inputs:

No inputs

### Results:

name	phone	age
Will Smith	666-666-6666	35
Bill Smith	555-555-1212	21
Phil Smith	222-555-5555	12

Now create a Mashup called PeopleGridMashup. It contains only a single grid widget. Add a reference to the InfoTableTestThing Service GetPeople. Make it Mashup Loaded.

Entity Type	Entity Name	Service	Mashup Loaded?	Remove
Things	InfoTableTestThing	GetPeople	<input checked="" type="checkbox"/>	<input type="button" value="X"/>

Now Bind All Data to the grid by dragging and dropping it.

**Drag All Data from Here**

**To Here and Bind to Data**

Save and view the Mashup.

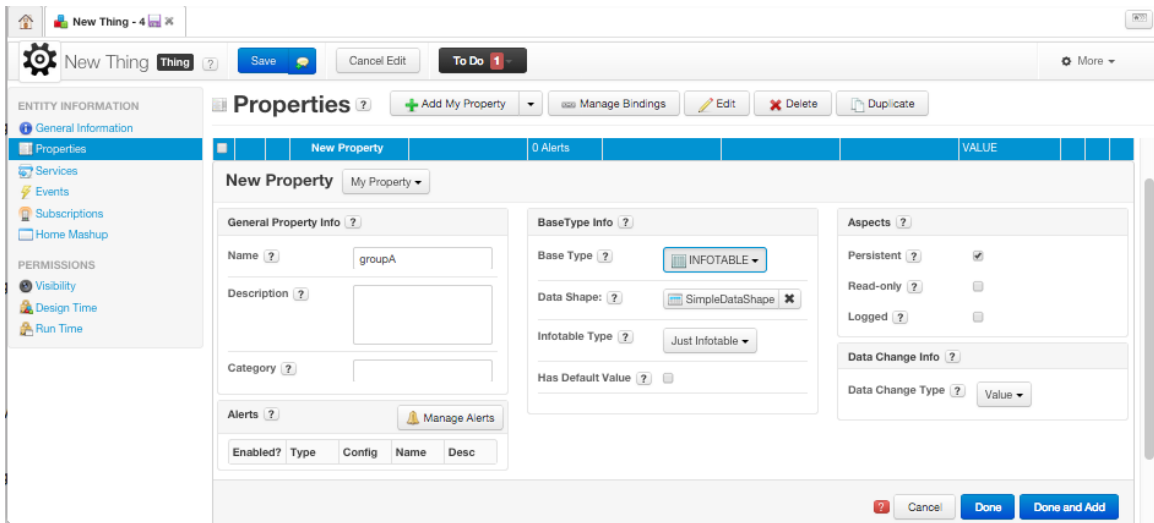
Name	Phone	Age
Will Smith	666-666-6666	35.00
Bill Smith	555-555-1212	21.00
Phil Smith	222-555-5555	12.00

## Declaring a Thing Property of Base Type InfoTable

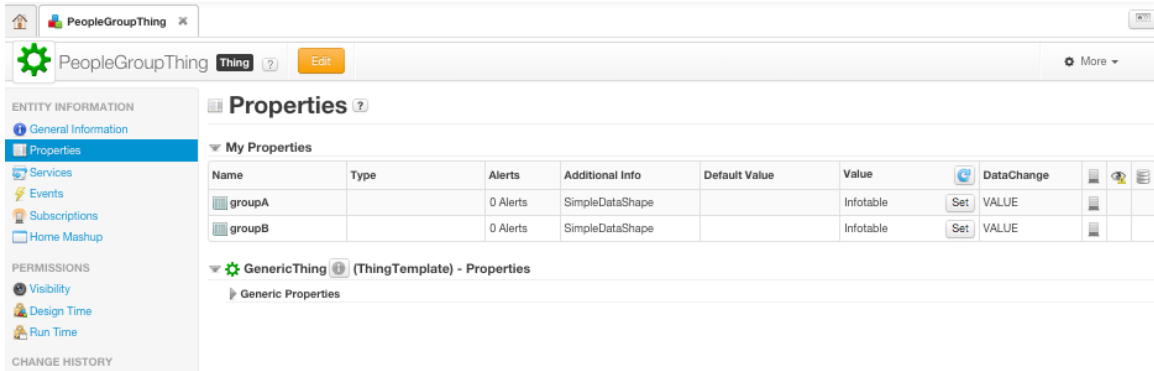
So far, we have been creating InfoTables from scratch but if we want them to be persistent, they must be stored either in a Stream, DataTable or as the Property of A Thing. Since Streams and DataTables are really a topic for their own discussion, let's create a thing that uses an InfoTable as one of its properties. Create the PeopleGroupThing shown below.

The screenshot shows the 'General Information' tab of a 'New Thing' editor. The 'Name' field is set to 'PeopleGroupThing'. The 'Description' field is empty. The 'Tags' field contains 'Search ModeTags or +'. The 'Thing Template' is set to 'GenericThing'. The 'Implemented Shapes' field contains 'Search ThingShapes or +'. On the right side, the 'Active' checkbox is checked. The 'Home Mashup' field contains 'Search Mashups or + to'. The 'Avatar' field has a 'Change' button. The 'Published' checkbox is unchecked. The 'Identifier' field has a 'Browse...' button. The 'Last Modified Date' field is set to 'No date and time selected'. The 'Value Stream' field contains 'Search Things or + to cre'.

Now declare two groups, groupA and groupB as properties of PeopleGroupThing and make them Persistable and make sure to specify the DataShape SimpleDataShape.



Then make an identical property called groupB. When done your properties should look like this.



Now lets add some entries to these InfoTables, this time using the Composer. Push the Set button on groupA, above. You will see the form below. The Add button will allow you to add as many new entries as you like.



Set value of property: groupA

	name	phone	age
<input type="checkbox"/>	<input type="text" value="Will Smith"/>	<input type="text" value="555-555-5555"/>	<input type="text" value="21"/>
<input type="checkbox"/>	<input type="text" value="Phil Smith"/>	<input type="text" value="222-222-5555"/>	<input type="text" value="32"/>

Cancel Set

And now add one person to groupB.

Set value of property: groupB

	name	phone	age
<input type="checkbox"/>	<input type="text" value="John Smith"/>	<input type="text" value="215-666-5555"/>	<input type="text" value="50"/>

Cancel Set

These InfoTables will be automatically persisted and can be manipulated using all the methods we have discussed above and new ones we will talk about below.

### Operations that can be performed on InfoTables

Once you have data in your InfoTables they can be easily searched, filtered or merged using operations in the InfoTable API. All the examples below are written as if they were Services in the previously created thing so that `me.groupA` and `me.groupB` already exist as persisted InfoTables.

### Sorting An InfoTable by any Column

```
// Create a new InfoTable containing all rows
// But now reordered Alphabetically by the name property
var result = Resources["InfoTableFunctions"].Sort(
    {
        sortColumn: "name",
        t: me.groupA,
        ascending: true
    }
);
```

### Filtering An InfoTable by any Column

```
// Create a new InfoTable containing only rows
// whoes name property is exactly "Will Smith"
var result = Resources["InfoTableFunctions"].EQFilter({
    t: me.groupA ,
    value: "Will Smith" ,
    fieldName: "name"
});
// One row will be in the new InfoTable
```

Rather than create an example for each kind filter function available, here is a list you can try out yourself with explanations.

Filter Function	Description
<b>GEFilter</b>	Filters source InfoTable for a column values greater than or equal to a value.
<b>GTFilter</b>	Filters source InfoTable for a column values greater than a value.
<b>LEFilter</b>	Filters source InfoTable for a column values less than or equal to a value.
<b>LikeFilter</b>	Filters by a Like Criteria Pattern
<b>LTFilter</b>	Filters source InfoTable for a column values less than a value.
<b>MissingValueFilter</b>	Filters by columns in which rows have missing or undefined values
<b>NearFilter</b>	A Geographic Radius Filter that operates on Columns of the Base Type "Location"
<b>NEFilter</b>	Finds all rows where a column is not equal to a value.
<b>RegexFilter</b>	Performs a filter on a column by applying a rule written in the Regular Expression Language. See <a href="http://en.wikipedia.org/wiki/Regular_expression">http://en.wikipedia.org/wiki/Regular_expression</a>
<b>SetFilter</b>	Same as the EQFilter except the criteria can be a comma separated list of matching values.

**TagFilter**

Searches the Tags applied to each row of the info table and filters out those that do not match

### *Querying for Rows in an InfoTable*

An InfoTable query allows you to perform multiple sorts and filters at the same time by constructing a query object as your criteria. We don't have an InfoTable large enough to perform a good demonstration of this feature but lets assume that me.groupA contains 100+ people in it and we want to write a Service that finds all members younger than age 20 and sort the results by age.

```
var query = {
  sorts: [
    {
      fieldName: "age",
      isAscending: false
    }
  ],
  filters: {
    type: "AND",
    filters: [
      {
        type: "LT",
        fieldName: "age",
        value: 20
      }
    ]
  }
};
var result = Resources['InfoTableFunctions'].Query(
  {
    query : query,
    t : me.groupA
  }
);
```

Note that the properties sorts and filters are arrays and could contain more than one criteria as well. Additional objects in the sorts array will influence sort order by sorting values that are not unique according to the previous sort criteria. Multiple filter criteria in the filters array would Unions (or) or Intersections (and) of the results of their individual filtered InfoTables. More on filtering with sets will be covered in the next section.

### *Performing Set Operations on InfoTables*

Set operations involve processing, comparing, extracting and combining one or more InfoTables to produce a new result.

Lets perform a Union between me.groupA and me.groupB which will produce a new InfoTable containing all Members.

```
// The joining or union of two InfoTables
var result = Resources["InfoTableFunctions"].Union(
    {
        t2: me.groupB ,
        t1: me.groupA
    }
);
// result now contains three rows or all the rows
// from groupA and groupB
```

Other useful set operations are...

Set Function	Description
<b>Distinct</b>	Removes duplicate rows based on the value of a column
<b>Interpolate</b>	Performs a statistical interpolation on time based data producing a new InfoTable by sampling the original InfoTable
<b>Intersect</b>	If given two InfoTables, produces a new InfoTable containing only the rows that are present in both.
<b>Pivot</b>	Creates a Pivot table. See <a href="http://en.wikipedia.org/wiki/Pivot_table">http://en.wikipedia.org/wiki/Pivot_table</a> for more information
<b>TopN</b>	Returns the Top N rows of the InfoTable or N most recently added rows of the InfoTable
<b>Union</b>	As demonstrated above, combines two InfoTables based on a common DataShape
<b>TimeShift</b>	Shifts all the specified timestamp fields in the InfoTable backwards or ahead by seconds.


### *Creating Calculated columns in InfoTables*

The DeriveFields function can be used to create calculated columns much in the same way a spreadsheet can produce new columns whose values are the result of a calculation performed on other row values.

```
var result = Resources["InfoTableFunctions"].DeriveFields(
    {
        t: me.groupA /* INFOTABLE */,
        columns: "TenTimesAge" /* STRING */,
        types: "NUMBER" /* STRING */,
        expressions: "age*10" /* STRING */
    }
);
```

Here is a screenshot of the resulting InfoTable when this service is run.

### derivedExample - Test Service

 Please be careful. Only execute services and queries where you understand the impacts.  
*Note that this is testing the currently saved script - any changes you have made since you have saved t*

**Inputs:**  
No inputs

**Results:**

TenTimesAge	name	phone	age
210	Will Smith	555-555-5555	21
320	Phil Smith	222-222-5555	32

## Conclusion

Now you know a lot more about InfoTables and how they can be created and used. InfoTables are the Data Structure used to insert and extract information from DataTables, Streams and ThingWorx's own APIs for configuration of Bindings, Security and Taging as well. They are also used to provide information about how a table should be rendered as an HTML document. Understanding InfoTables will enhance your ability to use other API calls in your projects and understand Services that are built on top of InfoTables as well.