Com’X 510 Custom Web Pages

7EN72-0199-01

11/2016

Safety information

Important information

Read these instructions carefully and look at the equipment to become familiar with the device before trying to install, operate, service or maintain it. The following special messages may appear throughout this manual or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

The addition of either symbol to a "Danger" or "Warning" safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.

This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

⚠ DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

⚠ WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

⚠ CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result in** minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

Please note

Electrical equipment should be installed, operated, serviced and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction, installation, and operation of electrical equipment and has received safety training to recognize and avoid the hazards involved.

Notices

This product must be installed, connected and used in compliance with prevailing standards and/or installation regulations. As standards, specifications and designs change from time to time, always ask for confirmation of the information given in this publication.

Schneider Electric
35 Rue Joseph Monier
92500 Rueil Malmaison – France
www.schneider-electric.com

Table of Contents

Safety Precautions

⚠ WARNING

INACCURATE DATA RESULTS

- Do not incorrectly configure the software, as this can lead to inaccurate reports and/or data results.
- Do not base your maintenance or service actions solely on messages and information displayed by the software.
- Do not rely solely on data displayed in the software to determine if the device is functioning correctly or meeting all applicable standards and requirements.
- Do not use data displayed in the software as a substitute for proper workplace practices or equipment maintenance.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

NOTICE

IMPAIRED COM'X PERFORMANCE

Do not add more than 10 MB of files to Custom Page Management.

Failure to follow these instructions can impair Com'X performance.

Introduction

This document describes the files you will need to create a custom web application on the Com'X 510. It also describes the available APIs, which allow you to:

- Request real-time values from devices
- Request historic samples from the Com'X 510

There are no device write capabilities within a custom web application. APIs are described in *Appendix A: API Reference*.

Prerequisites

To create custom web pages for the Com'X 510, the following prerequisites are necessary:

- A working knowledge of Modbus (for quantities that are not provided by the Com'X).
- A general understanding of the Internet and the World Wide Web (WWW).
- A working knowledge of HyperText Markup Language (HTML) and JavaScript.

Additional information

- *Com'X 510 User Manual*, DOCA0098EN
- *Converting an EGX300 Custom Page to a Com'X 510 Custom Page*, 7EN72-0200

Components of a Custom Page

Each custom page must have an initialization file (init.js) and an HTML file. A cascading stylesheet is optional but recommended.

Init.js

Init.js defines information about the application, such as where to find JavaScript, CSS, HTML, and image files. The init.js file is only valid if all the referenced files are uploaded to the Com'X.

This table describes components of the init.js file.

Item	Description
Application ID	Required. Internal identifier for the application. `APPLICATION_ID` must match the name you enter in the Com'X: **Custom Page Management** > **Create New Page** dialog.
Application title	Required. Title that displays on the application in Monitoring > Custom Pages.
HTML file	Required. This HTML snippet encapsulated in a container provides page layout.
JavaScript file(s)	Required. Includes calls to APIs. File list must match names of uploaded files.
CSS	Optional. If there is no CSS, the application inherits styles from the Com'X.
Images	Optional. Use to customize a page for specific use cases, for example, red and green LEDs, facility images, or buttons.

Below is a sample init.js file. The bold regions can be edited. All other syntax is required.

```
(function (OneESP) {
   var APPLICATION_ID = "demo";
      var _application =
OneESP.applicationsManager.getApplication(APPLICATION_
ID);
   var _prefix = "applications/" + APPLICATION_ID;
   var _sys_prefix = "/system/http";
   _application.prepare(
      //Set the application title
      "Com'X API Demo",
      //Set the HTML Content file
      _prefix + "/demo.html",
      //Set the required JS files
      [
         _sys_prefix + "/js/API_ComX_General.js",
         _prefix + "/demo.js"
      ],
      //Set the required CSS files
      [
         _prefix + "/demo.css"
      ]
   );
      // set optional image file(s)
      _application.contentImages = [
            _prefix + "/demoImage1.png",
            _prefix + "/demoImage2.png"
      ];
}( window.OneESP));
```

HTML File

The HTML page is formatted as a container. The ID for the must be unique across all applications. The rest of the content will depend on the desired appearance. Below is a sample .html file.

```
<div id="demo">

   <div>Com'X API Demo</div>

   <button
onclick="_ComXAPIDemoViewModel.listDevices()">List
Devices</button>
   <div id="content"></div>

</div>
```

JavaScript File(s)

Below is a sample JavaScript file that allows you to choose measurements per device and display those values in a table. The bold regions are Com'X APIs, described in *Appendix A: API Reference*.

NOTE: window.onload does not fire when the custom page is loaded, since the Com'X page is not a complete web page. Instead, _application.show gets called when the page is shown after initial load.

```
var _applicationsManager = OneESP.applicationsManager;

var ComXAPIDemoViewModel = function() {

   //Name of this specific application
   var APPLICATION_ID = "Demo";

   var myDevices;
   var deviceElements;
   var deviceElementNames;
   var selectedElementsWithIds = [];

   var _listDevices = function() {
      API_ComX_General.loadDevices(APPLICATION_ID,
function(data) {

         clearElement("content");

         myDevices = API_ComX_General.getDevices();
         if (myDevices.length > 0) {


   document.getElementById("content").appendChild(make
DeviceSelector("deviceList"));

   document.getElementById("content").appendChild(make
Button("List Elements", "listElements",
listElements));

         } else {
            alert("No Devices");
         }

      }, function(data) {
         // This is called if loadDevices fails.
         alert("Load Devices failed");
      });
   };

   return {
      listDevices: _listDevices
```

```
   };

   function clearElement(elementId) {
      document.getElementById(elementId).innerHTML =
"";
   }

   function makeButton(buttonText, id, onclickHandler)
{
      var button = document.createElement("button");
      button.innerHTML = buttonText;
      button.setAttribute("id", id);
      button.addEventListener("click", onclickHandler);
      return button;
   }

   function deviceListChange() {
      removeElement("content", "deviceElements");
      removeElement("content", "getElements");
      removeElement("content", "elementTable");
   }

   function makeDeviceSelector(id) {

      var selectElement =
document.createElement("select");

      for(var i = 0; i < myDevices.length; i++) {

         var option = new
Option(myDevices[i].deviceName, i);

   selectElement.options[selectElement.options.length]
= option;
      }
      selectElement.setAttribute("id", id);
      selectElement.addEventListener("change",
deviceListChange);

      return selectElement;
   }

   function removeElement(parentElement,
childElement){
     if ((childElement != parentElement) &&
(document.getElementById(childElement))) {
          var child =
document.getElementById(childElement);
          var parent =
document.getElementById(parentElement);
```

```
            parent.removeChild(child);
         }
      }

      function listElements() {
         removeElement("content", "deviceElements");
         removeElement("content", "getElements");
         removeElement("content", "elementTable");

         var deviceSelect =
document.getElementById("deviceList");
         var selectedOption = deviceSelect.selectedIndex;
         var selectedDevice =
deviceSelect.options[selectedOption].text;
         deviceElements =
API_ComX_General.getDeviceTopics(selectedDevice);
         deviceElementNames =
API_ComX_General.getDeviceTopicNames(selectedDevice);

         var selectElement =
document.createElement("select");

         for (var i = 0; i < deviceElements.length; i++) {
            var option = new Option(deviceElementNames[i],
deviceElements[i].element);

      selectElement.options[selectElement.options.length]
= option;
         }

         selectElement.setAttribute("id",
"deviceElements");
         selectElement.setAttribute("multiple",
"multiple");


      document.getElementById("content").appendChild(sele
ctElement);

      document.getElementById("content").appendChild(make
Button("Get Elements", "getElements", getElements));
      }

      function getElements() {
         var deviceSelect =
document.getElementById("deviceList");
         var selectedOption = deviceSelect.selectedIndex;
         var selectedDevice =
deviceSelect.options[selectedOption].text;
```

```
      var elementSelect =
document.getElementById("deviceElements");
      var selectedElements = [];
      selectedElementsWithIds = [];

      for (var i = 0; i < elementSelect.length; i++) {
         if(elementSelect.options[i].selected) {

   selectedElements.push(elementSelect.options[i].valu
e);

   selectedElementsWithIds.push({shortName:elementSele
ct.options[i].value, name:deviceElementNames[i]});
         }
      }


   API_ComX_General.getRealTimeSample(APPLICATION_ID,
selectedDevice, selectedElements, showElements);
   }

   function showElements(data) {
      removeElement("content", "elementTable");

      // Get returned device name
      var deviceIds = Object.keys(data);
      var deviceName =
API_ComX_General.getDeviceById(deviceIds[0]);

      // Create Element Table
      var elementTable =
document.createElement("table");
      elementTable.setAttribute("id", "elementTable");

      // Create Table Head
      var tableHead = document.createElement("thead");
      elementTable.appendChild(tableHead);

      var tableHeadRow = document.createElement("tr");
      tableHead.appendChild(tableHeadRow);

      var tableHeadCell = document.createElement("th");
      tableHeadRow.appendChild(tableHeadCell);

      // Add device name to the table head

   tableHeadCell.appendChild(document.createTextNode(d
eviceName.deviceName));
      tableHeadCell.colSpan = 2;
```

```
      // Table Column Titles
      var columnTitleRow =
document.createElement("tr");


   columnTitleRow.appendChild(document.createElement("
th"));

   columnTitleRow.appendChild(document.createElement("
th"));


   columnTitleRow.cells[0].appendChild(document.create
TextNode("Parameter"));

   columnTitleRow.cells[1].appendChild(document.create
TextNode("Value"));

      tableHead.appendChild(columnTitleRow);

      // Create table body
      var tableBody = document.createElement("tbody");

      // Add elements and values
      var len = Object.keys(data[deviceIds[0]]).length;
      for(var i = 0; i < len; i++) {
         var tbodyRow = document.createElement("tr");


   tbodyRow.appendChild(document.createElement("td"));

   tbodyRow.appendChild(document.createElement("td"));


   tbodyRow.cells[0].appendChild(document.createTextNo
de(selectedElementsWithIds[i].name));

   tbodyRow.cells[1].appendChild(document.createTextNo
de(data[deviceIds[0]][selectedElementsWithIds[i].short
Name].value + " " +

   (data[deviceIds[0]][selectedElementsWithIds[i].shor
tName].unit.symbol !== undefined ?

data[deviceIds[0]][selectedElementsWithIds[i].shortNam
e].unit.symbol : "")));

         tableBody.appendChild(tbodyRow);
      }
      elementTable.appendChild(tableBody);
```

```
   document.getElementById("content").appendChild(elem
entTable);

   }
};

//Create an instance of the view model
var _ComXAPIDemoViewModel = new
ComXAPIDemoViewModel();
```

CSS File

A CSS is optional but recommended. If there is no CSS, the application inherits styles from the Com'X. Below is an example CSS.

```
#elementTable {
   display: table;
   border: 2px solid black;
   border-collapse: collapse;
   vertical-align: middle;
}

#elementTable th {
   text-align: center;
   border: 1px solid black;
   padding: 2px;
}

#elementTable td {
   border: 1px solid black;
   padding: 5px;
}

#deviceList {
   position: absolute;
   top: 20px;
   left: 5px;
   width: 100px;
}

#listElements {
   position: absolute;
   top: 50px;
   left: 0px;
}

#content {
   position: absolute;
   top: 0px;
   left: 150px;
```

```
   display: inline;
   width: 600px;
}

#deviceElements {
   position: absolute;
   top: 20px;
   left: 120px;
   width: 200px;
}

#getElements {
   position: absolute;
   top: 20px;
   left: 330px;
}

#elementTable{
   position: absolute;
   top: 100px;
   left: 50px;

}
```

Appendix A: API Reference

Below are the functions related to real time data, historical data, and files.

Real Time Data

loadDevices (applicationID, fnDone, fnError, fnAlways, context)

Description	
Retrieves the list of devices and the elements they support from the Com'x and loads them into local memory. This must be called before using other device related functions	
Parameters	
applicationID	Mandatory. The application ID defined in the init.js file.
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object.	

getDevices ( )

Description
Returns an array of device objects in the following format: **deviceId**: internal resource id **deviceName**: device name, as defined in the Device Settings Table **elements**: array of elements defined for the device **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol **modbusId**: local Modbus device ID
Parameters
None
Returns
An array of device objects

getDeviceNames ( )

Description
Returns a list of device names as strings.
Parameters
None
Returns
An array of device name strings

getDeviceTopics (deviceName)

Description	
Returns a list of device topics as the following object: **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol	
Parameters	
deviceName	The name of the device from which to retrieve the device topics
Returns	
An array of device topic objects	

getDeviceTopicNames (deviceName)

Description	
Returns a list of human readable device topic names in the current user's language	
Parameters	
deviceName	The name of the device from which to retrieve the device topics names
Returns	
An array of device topics as Strings	

getDeviceTopicNamesById (deviceId)

Description	
Returns an array of human readable topic names supported by the device given the Com'x device ID	
Parameters	
deviceId	internal device ID
Returns	
An array of device topic names	

getDeviceTopicNamesByModbusId (modbusId)

Description	
Returns an array of human readable topic names supported by the device given the Modbus ID	
Parameters	
modbusId	local Modbus device ID
Returns	
An array of device topic names	

getDeviceTopicsById (deviceName)

Description	
Returns an array of topics as device topic objects supported by the device given the Com'X device ID. **deviceId**: internal resource id **deviceName**: device name, as defined in the Device Settings Table **elements**: array of elements defined for this device **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol **modbusId**: local Modbus device ID	
Parameters	
deviceName	The name of the device from which to retrieve the device topics names
Returns	
An array of device topic objects	

getDeviceByModbusId (modbusId)

Description
Returns device information as the following object: **deviceId**: internal resource id **deviceName**: device name, as defined in the Device Settings Table **elements**: array of elements defined for this device **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol **modbusId**: local Modbus device ID

Parameters	
modbusId	local Modbus device ID

Returns
A device object

getDeviceTopicsByModbusId (modbusId)

Description
Returns an array of topics as Short Name ID supported by the device given the device Modbus ID

Parameters	
modbusId	local Modbus device ID

Returns
An array of device topics

getDeviceById(deviceId)

Description
Returns device information as the following object: **deviceId**: internal resource id **deviceName**: device name, as defined in the Device Settings Table **elements**: array of elements defined for this device **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol **modbusId**: local Modbus device ID
Parameters
deviceId — internal device ID
Returns
A device object

getDeviceNameById(deviceId)

Description
Returns a device name string
Parameters
deviceId — internal device ID
Returns
Device name string

getDeviceNameByModbusId(modbusId)

Description
Returns a device name string
Parameters
modbusId — local Modbus device ID
Returns
Device name string

getRealTimeSample (applicationID, deviceName, element, fnDone, fnError, fnAlways, context)

Description
Gets real time data samples for the given device and element(s) in the following format: **element**: **timestamp**: *seconds* **unit**: **isSi**: true or false **name**: Name of unit **quantity**: name of quantity **symbol**: unit abbreviation **value**: data sample value

Parameters	
applicationID	Mandatory. The application ID defined in the init.js file.
deviceName	device name(s), as defined in the Device Settings Table. This can be a single device name, a comma-separated string of device names, or an array of device names.
element	internal Short Name ID. This can be a single element ID, a comma-separated string of element IDs, or an array of element IDs
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.

Returns
The jQuery XMLHttpRequest (jqXHR) object

getRealTimeSampleByModbusId (applicationID, modbusId, element, fnDone, fnError, fnAlways, context)

Description	
Gets real time data samples for the given device Modbus ID and element(s) in the following format.	
Parameters	
applicationID	Mandatory. The application ID defined in the init.js file.
modbusId	Local Modbus device ID. This can be a single Modbus ID, a comma-separated string of Modbus IDs, or an array of Modbus IDs.
element	Internal Short Name ID. This can be a single element ID, a comma-separated string of element IDs, or an array of element IDs.
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object	

Historical Device Data

loadHistoricalDevices (applicationID, fnDone, fnError, fnAlways, context)

Description	
Retrieves the list of devices that have historical data from the Com'x and loads them into local memory. This must be called before using other historical device-related functions.	
Parameters	
applicationID	The application ID defined in the init.js file. Mandatory.
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object.	

getHistoricalDevices ( )

Description

Returns a list of devices that have historical data as the following object:
 	
deviceId: internal resource id
deviceName: device name, as defined in the Device Settings Table
elements: array of element objects for elements that have historical data
  AccumulationType (optional): "Cumulative" or "Interval"
  element: internal Short Name ID
  metadata:
  name: unit name
  quantity: description of the quantity
  symbol: unit symbol
modbusId: local Modbus device ID

Parameters
None
Returns
List of device objects

getHistoricalDeviceNames ( )

Description
Returns a list of device names as strings for devices that have historical data.
Parameters
None
Returns
List of devices names

getHistoricalDeviceTopics (deviceName)

Description	

Returns a list of device topics that contain historical data as the following object:
 	
AccumulationType (optional): "Cumulative" or "Interval"
element: internal Short Name ID
metadata:
  name: unit name
  quantity: description of the quantity
  symbol: unit symbol	

Parameters	
deviceName	The name of the device from which to retrieve the device topics
Returns	

List of device topics

getHistoricalDeviceTopicNames (deviceName)

Description	
Returns a list of human readable device topic names in the current user's language	
Parameters	
deviceName	The name of the device that has historical data from which to retrieve the device topics names that have historical data
Returns	
List of device topic names	

getHistoricalDeviceByModbusId (modbusId)

Description	
Returns historical device information as the following object: **deviceId**: internal resource id **deviceName**: device name, as defined in the Device Settings Table **elements**: array of elements with historical data for this device **AccumulationType (optional)**: "Cumulative" or "Interval" **element**: internal Short Name ID **metadata**: **name**: unit name **quantity**: description of the quantity **symbol**: unit symbol **modbusId**: local Modbus device ID	
Parameters	
modbusId	local Modbus device ID
Returns	
Historical device object	

getHistoricalSamples (applicationID, deviceName, topic, start, end, fnDone, fnError, fnAlways, context)

Description	
Returns historical data samples for the given Device Name, Topic Name, and time range in the following format: **Date in Seconds**: *value*	
Parameters	
applicationID	Mandatory. The application ID defined in the init.js file.
deviceName	device name, as defined in the Device Settings Table
topic	internal Short Name ID
start	JavaScript Date object with the starting date and time
end	JavaScript Date object with the ending date and time
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object	

getHistoricalSamplesByModbusId (applicationID, modbusId, topic, start, end, fnDone, fnError, fnAlways, context)

Description	
Returns historical data samples for the given time range given the Modbus ID of the device and the Topic Name in the following format: **Date in Seconds**: *value*	
Parameters	
applicationID	Mandatory. The application ID defined in the init.js file.
modbusId	the Modbus ID of the Device
topic	internal Short Name ID
start	JavaScript Date object with the starting date and time
end	JavaScript Date object with the ending date and time
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	Optional. The object that invoked the function call.
Parameters	
The jQuery XMLHttpRequest (jqXHR) object	

Files

listFiles (applicationName, fnDone, fnError, fnAlways, context)

Description	
Returns a list of files in the application directory	
Parameters	
applicationName	The application ID defined in the init.js file.
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object	

storeFile (content, applicationName, fileName, fnDone, fnError, fnAlways, context)

Description	
Stores a file in your application directory.	
Parameters	
content	The content of the file
applicationName	The application ID defined in the init.js file.
fileName	The name of the file
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object	

getFile (applicationName, fileName, fnDone, fnError, fnAlways, context)

Description	
Retrieves file from your application directory.	
Parameters	
applicationName	The application ID defined in the init.js file.
fileName	The name of the file
fnDone	Optional. The function to call when the operation successfully completes.
fnError	Optional. The function to call when the operation encounters an error.
fnAlways	Optional. The function to call when the operation terminates.
context	The object that invoked the function call.
Returns	
The jQuery XMLHttpRequest (jqXHR) object	

Appendix B: PowerLogic Tags

PowerLogic tags are for legacy support and for elements that are not already defined in the Com'X. For more information on how PL tags are used in EGX300 custom web pages, see *PowerLogic™ Ethernet Gateway EGX300 Reference Guide*, 63230-319-230.

The Com'X creates a request from its devices using information contained in a PL tag. These tags indicate what type of request is desired (e.g. Read Holding Register), the device ID, and data supporting the request. The delimiter at the beginning of a tag is (PL__) and the delimiter at the end is (__PL). The table below lists the supported PL tags.

Function Name	Function Code	PowerLogic Tag
Modbus Block Read – Coil Status	Modbus Function Code 1	<DeviceID>^C<StartingCoilAddress>[<NumberOfCoils>] example tag = PL__1^C1003[5]__PL example of data returned = 1,0,0,1,1
Modbus Block Read – Input Status	Modbus Function Code 2	<DeviceID>^D<StartingInputAddress>[<NumberOfInputs>] example tag = PL__1^D1003[5]__PL example of data returned = 1,0,0,1,1
Modbus Block Read – Holding Registers	Modbus Function Code 3	<DeviceID>^H<StartingRegisterAddress>[<NumberOfRegisters>] example tag = PL__1^H1003[5]__PL example of data returned = 85,86,84,25,56
Modbus Block Read – Input Registers	Modbus Function Code 4	<DeviceID>^I<StartingRegisterAddress>[<NumberOfRegisters>] example tag = PL__1^I1003[5]__PL example of data returned = 85,86,84,25,56
Modbus Block Read – General Reference	Modbus Function Code 20	<DeviceID>^F<StartingRegisterAddress>,[<NumberOfRegisters>]<File> example tag = PL__1^F1003[5]2__PL example of data returned = 85,86,84,25,56
Modbus Scattered Read – Holding Registers	Modbus Function Code 100	<DeviceID>^S<RegisterAddress1>,<RegisterAddress2>,etc example tag = PL__1^S1003,1004,1005,1006,1007__PL example of data returned = 85,86,84,25,56