L5 Connect API Introduction

Last updated by | Fly, David C | Feb 20, 2025 at 3:08 PM CST

Initial Document Date 10/24/2023
Software Release 9.8.2.x

Contents

Goal

The purpose of this document is to explain the basic layout of the L5 Connect™ API and how it can be used to get information required to integrate the L5 Connect system with a customer designed interface.

Introduction

The L5 Connect™ system is built on top of a carefully designed relational database to provide data integrity, flexibility, and extendibility. The API reflects this design in the layout of the objects it provides for reading and updating. This document will help to explain the design philosophy behind the layout of the API and the typical use case for how customers manage data through the API.

Design Philosophy

There is a lot of data available for a user to retrieve and manage through the API. The objects available in the L5 API have attributes, which are the data directly related to that object, and relationships, which point to another type of object related to the first object. The ID provided in that relationship can be used to pull the related object and get its detailed attributes. Using this design allows the user to only pull the extra data from

relationships as needed and helps to reduce the total data transferred. For example, if a user requested an issued tool with the tool ID of 100004 from the API, the command would look like this.

```
GET /IssuedTools/100004
```

The response he would look something like this.

```
{
  "data": {
    "type": "issuedtools",
    "id": "100004",
    "attributes": {},
    "relationships": {
      "instances": {
        "data": [
          {
            "type": "issuedtoolInstance",
            "id": "100004",
            "attributes": {
              "issuetime": "2021-10-21T21:23:05.803",
              "intransit": false,
              "quantity": 1
            },
            "relationships": {
              "tool": {
                "data": {
                  "type": "tools",
                  "id": 100004
                }
              },
              "issuebehavior": {
                "data": {
                  "type": "issuebehaviors",
                  "id": 0
                }
              },
              "employee": {
                "data": {
                  "type": "employees",
                  "id": 5
                }
              },
```

```
            "location": {
              "data": {
                "type": "locations",
                "id": 35
              }
            }
          }
        }
      ]
    }
  }
}
```


This contains the attributes specific to an issued tool such as the time it was issued and the quantity issued. It also contains relationships to related objects in the system such as the type of tool that was issued and the person to whom it was issued.

If more information about the issuing user is required, a call can be made to the employees controller get by ID method using the ID provided in the relationship, which is 5 in this case. Here is what that request would look like.

```
GET /Employees/5
```

Here is an example of that response from the API.

```
{
  "data": {
    "type": "employees",
    "id": "5",
    "attributes": {
      "displayname": "Mechanic, Mike ",
      "imageupdated": "2022-05-16T20:13:09",
      "deactivated": false,
      "lastname": " Mechanic ",
      "firstname": "Mike",
      "middleinitial": "",
      "title": "",
      "username": "mjm",
      "badge": "9BACC9"
    },
    "relationships": {
      "homelocation": {
        "data": {
          "type": "locations",
          "id": 1
        }
      },
      "groups": {
        "data": [
          {
            "type": "groups",
            "id": 2
          }
        ]
      },
      "profiles": {
        "data": [
          {
            "type": "locations",
            "id": 1,
```

```
            "relationships": {
              "profile": {
                "data": {
                  "type": "profiles",
                  "id": 1
                }
              }
            }
          }
        ]
      }
    }
  }
}
```

This provides the attributes of the issuing employee such as his display name and user ID and more. The same approach can be used to get the tool specific information such as part number, description and other tool related fields.

Extended Filtering

Sometimes you might wish to filter by a field in a relationship to an object. To make certain cases like this more convenient, extended filtering has been added in select places. For instance, if you wanted to get all the eventlogs for an employee with a specific email, you could look at the eventlogs controller get and see that it supports extended filtering for the employee relationship. You could then look at the documentation of the employee controller and see that it supports extended filtering on the email parameter. You could then send the following request to get all the eventlogs for an employee with the specified email as an example. Notice how the relationship and the parameter desired are combined with a dash to create the parameter key to be requested.

```
GET /EventLogs?employee-email=johndoe@gmail.com
```

This would return a list of the eventlogs that are for the employee with an email of johndoe@gmail.com.

The documentation of the individual controllers in the API will define any extended filtering relationships including string search capabilities. They will also define any fields within that controller which can be used for extended filtering.

String Search Capability

String search capability has been added to certain parameters in the API to allow SQL LIKE string pattern matching type searches. For example, there is a description_search filter on the master tools get method that

allows you to search for master tools with a tool description that contains a defined string pattern. There are two types of wild cards that can be used to help build the string pattern you want to match.

- The percent sign % represents zero, one, or multiple characters
- The underscore sign _ represents one, single character

So, if you wanted to search for tools where the description contained the word wrench it would look like this.

```
GET /MasterTools?description_search=%wrench%
```

The % characters mean that any combination of characters in front of the string wrench and any combination of characters after the string wrench will be considered a match.

The documentation of the individual controllers in the API will define any string search capabilities.

Sample Use Cases

Issued Tool Monitoring using Event Logs

This section will describe the typical use case for monitoring issued tools through the API of the L5 Connect™ system. This is how most of our customers have set up their interfaces. In the L5 Connect™ system, eventlogs provide a record of everything that occurs in the system. There are many kinds of events recorded in the system. Each eventlog has an **action** relationship which contains a logactions ID. This ID represents a type of event that can occur in the L5 Connect™ system. To get a list of the logactions that are recorded in the system you can use the **GET** method on the **LogActions** controller. The request would look like this.

```
GET /LogActions
```

And here is a partial example of the response.

```
{
  "data": [
    {
      "type": "logactions",
      "id": "0",
      "attributes": {
        "en": "LOG ERROR",
        "it": "LOG ERRORE",
        "fr": "LOG ERREUR",
        "es": "ERROR LOG",
        "de": "ERROR LOG",
        "pt": "Log de erro",
        "zh": "日志错误",
        "ko": "로그 오류",
        "ja": "ログエラー"
      }
    },
    ....
    {
      "type": "logactions",
      "id": "6144",
      "attributes": {
        "en": "Tool Issued",
        "it": "prelevato strumento",
        "fr": "Outil Publié",
        "es": "Herramienta Emitido",
        "de": "Werkzeugausgabe",
        "pt": "Ferramenta Emitido",
        "zh": "工具借出",
        "ko": "공구 발급됨",
        "ja": "ツールの取り出し"
      }
    },
    {
      "type": "logactions",
      "id": "6145",
      "attributes": {
        "en": "Tool Returned",
        "it": "Strumento restituito",
        "fr": "Outil de retour de",
        "es": "Herramienta Obtenidos",
        "de": "Werkzeug Returned",
        "pt": "Ferramenta para Devolução",
        "zh": "工具归还",
        "ko": "공구 반환됨",
        "ja": "ツールの返却"
    },
    ....
    {
      "type": "logactions",
      "id": "2051",
      "attributes": {
        "en": "Drawer Closed",
        "it": "Cassetto Chiuso",
        "fr": "Tiroir fermé",
        "es": "Cajón Cerrado",
        "de": "Schublade geschlossen",
        "pt": "Gaveta Fechado",
        "zh": "抽屉关闭",
        "ko": "서랍 닫힘",
        "ja": "引き出しが閉じた状態"
      }
    }
  ],
  "links": {
    "first": "https://l5connectapi.com:59011/api/LogActions?offset=0&limit=50",
```

```
    "next": "https://l5connectapi.com:59011/api/LogActions?offset=50&limit=50",
    "last": "https://l5connectapi.com:59011/api/LogActions?offset=600&limit=50"
  },
  "meta": {
    "offset": 0,
    "limit": 50,
    "count": 620
  }
}
Response headers
 content-length: 22996
 content-type: application/json; charset=utf-8
Responses
```

For this example, we will be interested in the actions for each time a tool is issued or returned. You can see from the logactions sample that those logaction IDs are 6144 and 6145. Using the Eventlogs controller of the API, you can poll the API to get the latest group of eventlogs filtered for the action type of concern. In this case that would be the tool issued, and tool returned events. You could also filter by a specific location or device to limit your results further. See the Getting Eventlogs for a Specific Device section for more information.

Here is what that request might look like.

```
GET /Eventlogs?limit=2&actions=6144%2C%206145&startid=1&endid=500000
```

Here is an example of what the response might look like.

```
{
  "data": [
    {
      "type": "eventlogs",
      "id": "18596",
      "attributes": {
        "eventtime": "2020-09-28T19:17:22.577",
        "quantity": 2,
        "data": "2"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 6144
          }
        },
        "tool": {
          "data": {
            "type": "tools",
            "id": 100014
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 41
          }
        },
        "locationdest": {
          "data": {
            "type": "locations",
            "id": 35
          }
        },
```

```
    "parentsourcelocation": {
      "data": {
        "type": "locations",
        "id": 41
      }
    },
    "parentdest": {
      "data": {
        "type": "locations",
        "id": 21
      }
    },
    "employee": {
      "data": {
        "type": "employees",
        "id": 4
      }
    },
    "affectedemployee": {
      "data": {
        "type": "employees",
        "id": 4
      }
    },
    "device": {
      "data": {
        "type": "devices",
        "id": 41
      }
    }
  }
},
{
  "type": "eventlogs",
  "id": "78263",
```

```
"attributes": {
  "eventtime": "2020-11-03T16:27:14.147",
  "quantity": 1,
  "data": "1"
},
"relationships": {
  "action": {
    "data": {
      "type": "logactions",
      "id": 6144
    }
  },
  "tool": {
    "data": {
      "type": "tools",
      "id": 100004
    }
  },
  "locationsource": {
    "data": {
      "type": "locations",
      "id": 41
    }
  },
  "locationdest": {
    "data": {
      "type": "locations",
      "id": 35
    }
  },
  "parentsourcelocation": {
    "data": {
      "type": "locations",
      "id": 41
    }
```

```
      },
      "parentdest": {
        "data": {
          "type": "locations",
          "id": 21
        }
      },
      "employee": {
        "data": {
          "type": "employees",
          "id": 4
        }
      },
      "affectedemployee": {
        "data": {
          "type": "employees",
          "id": 4
        }
      },
      "device": {
        "data": {
          "type": "devices",
          "id": 41
        }
      }
    }
  }
],
"links": {
  "first": "http://localhost:59011/api/Eventlogs?limit=2&actions=6144%2c+6145&startid=1&endid=500000&offset=
  "next": "http://localhost:59011/api/Eventlogs?limit=2&actions=6144%2c+6145&startid=1&endid=500000&offset=2
  "last": "http://localhost:59011/api/Eventlogs?limit=2&actions=6144%2c+6145&startid=1&endid=500000&offset=5
},
"meta": {
  "offset": 0,
```

```
    "limit": 2,
    "count": 512
  }
}
```

Notice that most of the data is relationships. To make it easier to get the data from those relationships, the user can add includes. This tells the API to go ahead and get that information and add it to the data returned. For this example, we will add includes for the tool and the employee responsible for the tool. Here is the request with includes added.

```
GET /Eventlogs?limit=2&includes=employee%2C%20tool&actions=6144%2C%206145&startid=1&endid=500000
```

Here is what that result would look like with the added include data.

```
{
  "data": [
    {
      "type": "eventlogs",
      "id": "18596",
      "attributes": {
        "eventtime": "2020-09-28T19:17:22.577",
        "quantity": 2,
        "data": "2"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 6144
          }
        },
        "tool": {
          "data": {
            "type": "tools",
            "id": 100014
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 41
          }
        },
        "locationdest": {
          "data": {
            "type": "locations",
            "id": 35
          }
        },
```

```
    "parentsourcelocation": {
      "data": {
        "type": "locations",
        "id": 41
      }
    },
    "parentdest": {
      "data": {
        "type": "locations",
        "id": 21
      }
    },
    "employee": {
      "data": {
        "type": "employees",
        "id": 4
      }
    },
    "affectedemployee": {
      "data": {
        "type": "employees",
        "id": 4
      }
    },
    "device": {
      "data": {
        "type": "devices",
        "id": 41
      }
    }
  }
},
{
  "type": "eventlogs",
  "id": "78263",
```

```
"attributes": {
  "eventtime": "2020-11-03T16:27:14.147",
  "quantity": 1,
  "data": "1"
},
"relationships": {
  "action": {
    "data": {
      "type": "logactions",
      "id": 6144
    }
  },
  "tool": {
    "data": {
      "type": "tools",
      "id": 100004
    }
  },
  "locationsource": {
    "data": {
      "type": "locations",
      "id": 41
    }
  },
  "locationdest": {
    "data": {
      "type": "locations",
      "id": 35
    }
  },
  "parentsourcelocation": {
    "data": {
      "type": "locations",
      "id": 41
    }
```

```
      },
      "parentdest": {
        "data": {
          "type": "locations",
          "id": 21
        }
      },
      "employee": {
        "data": {
          "type": "employees",
          "id": 4
        }
      },
      "affectedemployee": {
        "data": {
          "type": "employees",
          "id": 4
        }
      },
      "device": {
        "data": {
          "type": "devices",
          "id": 41
        }
      }
    }
  }
],
"included": [
  {
    "type": "employees",
    "id": "4",
    "attributes": {
      "displayname": "Mechanic, Dave",
      "imageupdated": "2022-08-29T15:31:43",
```

```
    "deactivated": false,
    "lastname": "Mechanic",
    "firstname": "Dave",
    "middleinitial": "",
    "title": "",
    "customerid": "dvm",
    "username": "dvm",
    "email": "dvm123@gmail.com",
    "cellphone": "5018675309",
    "carrier": 2,
    "badge": "9BBB4B"
  },
  "relationships": {
    "homelocation": {
      "data": {
        "type": "locations",
        "id": 1
      }
    },
    "language": {
      "data": {
        "type": "languages",
        "id": 0
      }
    },
    "temporarybadge": {
      "data": {
        "type": "temporarybadges",
        "id": 4
      }
    },
    "profiles": {
      "data": [
        {
          "type": "locations",
```

```
          "id": 1,
          "relationships": {
            "profile": {
              "data": {
                "type": "profiles",
                "id": 1
              }
            }
          }
        },
        {
          "type": "locations",
          "id": 47,
          "relationships": {
            "profile": {
              "data": {
                "type": "profiles",
                "id": 34
              }
            }
          }
        }
      ]
    }
  }
},
{
  "type": "tools",
  "id": "100004",
  "attributes": {
    "deactivated": false,
    "customerid": "Torque!",
    "quantity": 1
  },
  "relationships": {
```

```
"mastertool": {
  "data": {
    "type": "mastertools",
    "id": 100014
  }
},
"defaulttool": {
  "data": {
    "type": "defaulttools",
    "id": 100014
  }
},
"homelocation": {
  "data": {
    "type": "locations",
    "id": 41
  }
},
"devicelocation": {
  "data": {
    "type": "locations",
    "id": 41
  }
},
"parenttool": {
  "data": {
    "type": "toolparentchild",
    "id": 100004,
    "relationships": {
      "parenttool": {
        "data": {
          "type": "tools",
          "id": 100115
        }
      },
```

```
          "childtool": {
            "data": {
              "type": "tools",
              "id": 100004
            }
          },
          "locationgeneric": {
            "data": {
              "type": "locationgenerics",
              "id": 5
            }
          }
        }
      }
    },
    "issuedtool": {
      "data": {
        "type": "issuedtools",
        "id": 100004
      }
    },
    "toolstatus": {
      "data": {
        "type": "toolstatuses",
        "id": 100004
      }
    },
    "toolmaintenances": {
      "data": {
        "type": "toolmaintenancesfortool",
        "id": 100004
      }
    }
  }
},
```

```
{
  "type": "tools",
  "id": "100014",
  "attributes": {
    "deactivated": false,
    "quantity": 8,
    "tag": "100014"
  },
  "relationships": {
    "mastertool": {
      "data": {
        "type": "mastertools",
        "id": 100024
      }
    },
    "defaulttool": {
      "data": {
        "type": "defaulttools",
        "id": 100024
      }
    },
    "homelocation": {
      "data": {
        "type": "locations",
        "id": 41
      }
    },
    "devicelocation": {
      "data": {
        "type": "locations",
        "id": 41
      }
    }
  }
}
```

```
  ],
  "links": {
    "first": "http://localhost:59011/api/Eventlogs?limit=2&includes=employee%2c+tool&actions=6144%2c+6145&star
    "next": "http://localhost:59011/api/Eventlogs?limit=2&includes=employee%2c+tool&actions=6144%2c+6145&start
    "last": "http://localhost:59011/api/Eventlogs?limit=2&includes=employee%2c+tool&actions=6144%2c+6145&start
  },
  "meta": {
    "offset": 0,
    "limit": 2,
    "count": 512
  }
}
```


This call could be customized as desired and further information can be gathered by additional calls for other relationship data as needed.

Getting Eventlogs for a Specific Device

You can get the eventlogs for a specific device in the system easily by adding a location ID filter to your request. Every device in the system is a location and has an associated location ID. To get the location ID for the device in which you are interested, you would get the list of locations in the system and look at the location data for the specific device. Here is what that request would look like.

```
GET /Locations
```

And here is an example of what the result might look like for one of the locations in the response.

```
},
{
  "type": "locations",
  "id": "3",  <=============== LOCATION ID
  "attributes": {
    "deactivated": false,
    "customerid": "Avionics Box 23",
    "name": "Avionics Box 23"
  },
  "relationships": {
    "locationtype": {
      "data": {
        "type": "locationtypes",
        "id": 3
      }
    },
    "parent": {
      "data": {
        "type": "locations",
        "id": 1
      }
    },
    "device": {
      "data": {
        "type": "devices",
        "id": 3
      }
    }
  }
},
```

You would look at the name or customerid to determine which location ID should be used for the filter. For example, the location ID for the Avionics Box 23 is 3. Now that you have the location ID of interest you can add the location ID filter to your eventlog request.

The result will be a list of eventlogs filtered by the location ID. Keep in mind that there are multiple location relationships in an eventlog. There are source and destination location, source and destination parent location, and parent group. Depending on the type of eventlog of interest, these may or may not be relevant. For this example, we will have eventlogs for the specified device.

Here is what that request would look like.

```
GET /Eventlogs?locationId=3
```

And here is a partial example of the response. All the eventlogs have a locationsource of 3.

```
{
  "data": [
    {
      "type": "eventlogs",
      "id": "1678",
      "attributes": {
        "eventtime": "2022-05-10T19:56:24.903",
        "data": "Z91FF876"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 2069
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 3
          }
        },
        "parentsourcelocation": {
          "data": {
            "type": "locations",
            "id": 1
          }
        },
        "employee": {
          "data": {
            "type": "employees",
            "id": 3
          }
        }
      }
    },
    {
      "type": "eventlogs",
      "id": "1679",
      "attributes": {
        "eventtime": "2022-05-10T19:56:24.903",
        "data": " -> 9.3.5.0"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 36889
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 3
          }
        },
        "parentsourcelocation": {
          "data": {
            "type": "locations",
            "id": 1
          }
        }
      }
    },

    ....

    {
      "type": "eventlogs",
```

```
      "id": "3221",
      "attributes": {
        "eventtime": "2022-05-10T18:58:10.463"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 8258
          }
        },
        "tool": {
          "data": {
            "type": "tools",
            "id": 100806
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 3
          }
        },
        "parentsourcelocation": {
          "data": {
            "type": "locations",
            "id": 1
          }
        },
        "employee": {
          "data": {
            "type": "employees",
            "id": 3
          }
        },
        "dataidwas": {
          "data": {
            "type": "mastertools",
            "id": 0
          }
        },
        "dataidis": {
          "data": {
            "type": "mastertools",
            "id": 100068
          }
        }
      }
    }
  ],
  "links": {
    "first": "https://l5connectapi.com:59011/api/Eventlogs?locationId=3&offset=0&limit=50",
    "next": "https://l5connectapi.com:59011/api/Eventlogs?locationId=3&offset=50&limit=50",
    "last": "https://l5connectapi.com:59011/api/Eventlogs?locationId=3&offset=600&limit=50"
  },
  "meta": {
    "offset": 0,
    "limit": 50,
    "count": 615
  }
}
```


Getting a Current List of Issued Tools

Let's say you wanted a way to get a list of the issued tools in the system and their related data other than monitoring tools issued from the event logs. The tools controller get method will allow you to get the

information you need with the proper filtering. Setting the **isIssued** filter to **true** will give you a list of all the tools issued in the system. Adding **"issuedtool"** to the **includes** filter of the request will also include a list of the issued tool relationship data for each of the issued tools in the **Included** part of the response.

The included issued tools information will contain IDs for related objects. For example, information about an employee ID can be obtained from the Employees controller to create a static cross-reference table when interfacing with other systems.

Here is what the get request with "isIssued = true" and "included = issuedtool" would look like.

```
GET /Tools?includes=issuedtool&isIssued=true
```

And here is an example of the response.

```
{
  "data": [
    {
      "type": "tools",
      "id": "100004",
      "attributes": {
        "deactivated": false,
        "customerid": "Torque!",
        "quantity": 1
      },
      "relationships": {
        "mastertool": {
          "data": {
            "type": "mastertools",
            "id": 100014
          }
        },
        "defaulttool": {
          "data": {
            "type": "defaulttools",
            "id": 100014
          }
        },
        "homelocation": {
          "data": {
            "type": "locations",
            "id": 41
          }
        },
        "devicelocation": {
          "data": {
            "type": "locations",
            "id": 41
          }
        },
        "parenttool": {
          "data": {
            "type": "toolparentchild",
            "id": 100004,
            "relationships": {
              "parenttool": {
                "data": {
                  "type": "tools",
                  "id": 100115
                }
              },
              "childtool": {
                "data": {
                  "type": "tools",
                  "id": 100004
                }
              },
              "locationgeneric": {
                "data": {
                  "type": "locationgenerics",
                  "id": 5
                }
              }
            }
          }
        },
        "issuedtool": {
          "data": {
            "type": "issuedtools",
            "id": 100004
          }
        },
        "toolstatus": {
          "data": {
```

```
          "type": "toolstatuses",
          "id": 100004
        }
      },
      "toolmaintenances": {
        "data": {
          "type": "toolmaintenancesfortool",
          "id": 100004
        }
      }
    }
  },
  {
    "type": "tools",
    "id": "100021",
    "attributes": {
      "deactivated": false,
      "quantity": 1
    },
    "relationships": {
      "mastertool": {
        "data": {
          "type": "mastertools",
          "id": 100028
        }
      },
      "defaulttool": {
        "data": {
          "type": "defaulttools",
          "id": 100028
        }
      },
      "homelocation": {
        "data": {
          "type": "locations",
          "id": 41
        }
      },
      "devicelocation": {
        "data": {
          "type": "locations",
          "id": 41
        }
      },
      "issuedtool": {
        "data": {
          "type": "issuedtools",
          "id": 100021
        }
      },
      "toolstatus": {
        "data": {
          "type": "toolstatuses",
          "id": 100021
        }
      }
    }
  },
  list of tools continues...
],
"included": [
  {
    "type": "issuedtools",
    "id": "100004",
    "attributes": {},
    "relationships": {
      "instances": {
        "data": [
          {
            "type": "issuedtoolInstance",
```

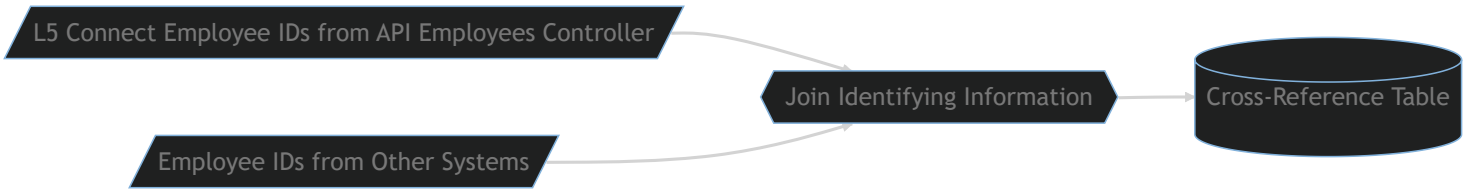
```
              "id": "100004",
              "attributes": {
                "issuetime": "2021-10-21T21:23:05.803",
                "intransit": false,
                "quantity": 1
              },
              "relationships": {
                "tool": {
                  "data": {
                    "type": "tools",
                    "id": 100004
                  }
                },
                "issuebehavior": {
                  "data": {
                    "type": "issuebehaviors",
                    "id": 0
                  }
                },
                "employee": {
                  "data": {
                    "type": "employees",
                    "id": 5   <---- HERE IS THE EMPLOYEE ID
                  }
                },
                "location": {
                  "data": {
                    "type": "locations",
                    "id": 35
                  }
                }
              }
            }
          ]
        }
      }
    },
    issued tool include data continues...
  ],
  "links": {
    "first": "http://localhost:59011/api/Tools?includes=issuedtool&isIssued=true&offset=0&limit=50",
    "next": "http://localhost:59011/api/Tools?includes=issuedtool&isIssued=true&offset=50&limit=50",
    "last": "http://localhost:59011/api/Tools?includes=issuedtool&isIssued=true&offset=100&limit=50"
  },
  "meta": {
    "offset": 0,
    "limit": 50,
    "count": 131
  }
}
```


Getting Issued Tools with Cross Referenced Employees

As of software release 9.8.4.1116, the L5 Connect API can pull a filtered list of issue tools that include the issued to employee identifying information. This identifying information can be used in a table that will “cross reference” employees in the L5 Connect system to employees in other systems.

Create Cross-Reference Table

This can be done once and refreshed when new employees are added. The table that is created is used in the “Normal Operation” process.



Normal Operation

The Issued Tool – Employee ID information can be pulled from the L5 Connect API on-demand. The cross-reference table that was previously created will be used to convert the L5 Connect Employee ID to the required identifier for other system(s). The attached document (Getting Current List of Issued Tools) provides detailed instructions for making the proper API request and interpreting the returned data to accomplish the data flow shown below.

NOTE: Additional details about the GET request and the returned data can the found in the Getting Current List of Issued Tools of the L5 Connect API Introduction document.

Managing Tool Maintenances

This example will show how you could use the API to manage your tool maintenances. Tools in the L5 Connect™ system have a master tool that defines the attributes that apply to all instances of that tool type (part number, description, etc.) and instances of that master tool which contain the attributes specific to that instance of a tool (tag, serial number, etc.). A master maintenance defines a type of maintenance that could be assigned to a master tool. The master maintenance defines the parts of a maintenance type that are common to all tool instances such as the maintenance period, the number of days before expiration where the system will display a warning, and the initial maintenance period when a tool instance is assigned a maintenance. A tool maintenance defines the attributes of the tool maintenance which are specific to that tool instance such as when the tool last received maintenance.

The L5 Connect™ system alerts users that a tool is about to need maintenance or is past due for maintenance by setting a status on that tool. The application of these statuses to the tool are recorded as eventlogs in the eventlog history. By monitoring the eventlogs for the **Status Set** action you can see that a status has been set on a tool. By using the log actions controller as described earlier in the document, you can get the status set action ID of 8204. The **dataidis** field, (data ID is) contains the status ID that was set for the tool. The status ID definitions can be found by using the GET on the StatusType controller. You can see that the ApiStatusStype class has the name as a relationship to a custom text field so we will want to include "name" in the includes filter for the request, which will look like this.

```
GET StatusType?includes=name
```

This will return a list of all the potential statuses that could be set in the system, with a list of custom text values for the names at the end. Scanning through the list you will see that the maintenance overdue status ID is 9 and the maintenance pending ID is 10.

```
{
  "data": [
    {    {
      "type": "statustypes",
      "id": "9",
      "attributes": {
        "usercustomizable": false,
        "background": "#FF0000",
        "generatewarning": true,
        "generatealert": true,
        "managedoutofbox": false,
        "toolboxtool": true,
        "toolbox": true,
        "rfidlockertool": true,
        "rfidlocker": true,
        "toolcribtool": true,
        "toolcrib": true,
        "kiosktool": true,
        "kiosk": true,
        "portaltool": true,
        "portal": true,
        "kiosktooldropoff": false
      },
      "relationships": {
        "name": {
          "data": {
            "type": "customtext",
            "id": 20
          }
        },
        "abbrev": {
          "data": {
            "type": "customtext",
            "id": 21
          }
        }
      }
    },
    {
      "type": "statustypes",
      "id": "10",
      "attributes": {
        "usercustomizable": false,
        "background": "#FFA500",
        "generatewarning": true,
        "generatealert": false,
        "managedoutofbox": false,
        "toolboxtool": true,
        "toolbox": true,
        "rfidlockertool": true,
        "rfidlocker": true,
        "toolcribtool": true,
        "toolcrib": true,
        "kiosktool": true,
        "kiosk": true,
        "portaltool": true,
        "portal": true,
        "kiosktooldropoff": false
      },
    ......
],
  "included": [
    {
      "type": "customtext",
      "id": "20",
      "attributes": {
        "en": "Maintenance Overdue",
        "it": "Manutenzione in ritardo",
        "fr": "Retard maintenance",
```

```
      "es": "Mantenimiento atrasado",
      "de": "Wartung überfällig",
      "pt": "Manutenção vencida",
      "zh": "维护期限已过",
      "ko": "유지 보수 기간이 지났습니다.",
      "ja": "メンテナンス期限切れ",
      "localized": true
    }
  },
  {
    "type": "customtext",
    "id": "22",
    "attributes": {
      "en": "Maintenance Pending",
      "it": "In attesa di manutenzione",
      "fr": "Maintenance en attente",
      "es": "Pendiente de mantenimiento",
      "de": "Wartung ausständig",
      "pt": "Manutenção pendente",
      "zh": "维护期限待审批",
      "ko": "유지 보수 승인중",
      "ja": "メンテナンス 未確定",
      "localized": true
    }
  },
```


If you wanted to monitor for tools that had a pending maintenance due, you would poll the eventlogs filtering for an action of 8204, which is the status set action. You would then look at the dataidis relationship for an ID of 10, which has a name relationship ID of 22, which is maintenance pending. So, your request would look like this.

```
GET /Eventlogs?actions=8204
```

And an example response of...

```
{
  "data": [
    {
      "type": "eventlogs",
      "id": "881521",
      "attributes": {
        "eventtime": "2024-04-24T20:44:34.747",
        "drawer": 8
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 8204
          }
        },
        "tool": {
          "data": {
            "type": "tools",
            "id": 225455
          }
        },
        "roi": {
          "data": {
            "type": "roi",
            "id": 100791
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 47
          }
        },
        "parentsourcelocation": {
          "data": {
            "type": "locations",
            "id": 1
          }
        },
        "device": {
          "data": {
            "type": "devices",
            "id": 47
          }
        },
        "dataidis": {
          "data": {
            "type": "statustypes",
            "id": 9
          }
        }
      }
    },
    {
      "type": "eventlogs",
      "id": "881526",
      "attributes": {
        "eventtime": "2024-04-24T20:46:49.403",
        "drawer": 8
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 8204
          }
        },
        "tool": {
```

```
      "data": {
        "type": "tools",
        "id": 225455
      }
    },
    "roi": {
      "data": {
        "type": "roi",
        "id": 100791
      }
    },
    "locationsource": {
      "data": {
        "type": "locations",
        "id": 47
      }
    },
    "parentsourcelocation": {
      "data": {
        "type": "locations",
        "id": 1
      }
    },
    "device": {
      "data": {
        "type": "devices",
        "id": 47
      }
    },
    "dataidis": {
      "data": {
        "type": "statustypes",
        "id": 10                <========== Here is our maintenance pending status ID
      }
    }
  }
},
{
  "type": "eventlogs",
  "id": "881527",
  "attributes": {
    "eventtime": "2024-04-24T21:04:33.477"
  },
  "relationships": {
    "action": {
      "data": {
        "type": "logactions",
        "id": 8204
      }
    },
    "tool": {
      "data": {
        "type": "tools",
        "id": 102458
      }
    },
    "locationsource": {
      "data": {
        "type": "locations",
        "id": 47
      }
    },
    "parentsourcelocation": {
      "data": {
        "type": "locations",
        "id": 1
      }
    },
    "device": {
      "data": {
```

```
            "type": "devices",
            "id": 47
          }
        },
        "dataidis": {
          "data": {
            "type": "statustypes",
            "id": 17
          }
        }
      }
    },
    {
      "type": "eventlogs",
      "id": "881531",
      "attributes": {
        "eventtime": "2024-04-25T16:11:09.15"
      },
      "relationships": {
        "action": {
          "data": {
            "type": "logactions",
            "id": 8204
          }
        },
        "tool": {
          "data": {
            "type": "tools",
            "id": 102458
          }
        },
        "locationsource": {
          "data": {
            "type": "locations",
            "id": 47
          }
        },
        "parentsourcelocation": {
          "data": {
            "type": "locations",
            "id": 1
          }
        },
        "device": {
          "data": {
            "type": "devices",
            "id": 47
          }
        },
        "dataidis": {
          "data": {
            "type": "statustypes",
            "id": 18
          }
        }
      }
    }
  ],
  "links": {
    "first": "http://localhost:59011/api/Eventlogs?offset=0&actions=8204&limit=50",
    "last": "http://localhost:59011/api/Eventlogs?offset=7300&actions=8204&limit=50"
  },
  "meta": {
    "offset": 7310,
    "limit": 50,
    "count": 7320
  }
}
```

So, we have identified a tool that has an impending maintenance that is due. Let's say that the tool was sent off to the calibration lab and has now been maintained and is ready to go back into service. And let's further assume you want to update the last maintenance date for the tool in the L5 Connect™ system to clear the status set against that tool. From the eventlog that told us there was a pending maintenance we can get the tool ID from the tool relationship. We also need to know the master maintenance ID of the maintenance type we will be updating, as a tool can have multiple maintenance types, and we need to update the correct one. You can use the GET method of the ToolMaintenancesForTool controller to get the list of maintenances associated with the tool.

```
GET /ToolMaintenancesForTool/225455
```


```
{
  "data": {
    "type": "toolmaintenancesfortool",
    "id": "225455",
    "attributes": {},
    "relationships": {
      "instances": {
        "data": [
          {
            "type": "toolmaintenanceinstance",
            "id": "225455",
            "attributes": {
              "datelastmaintenance": "2024-04-21T00:00:00",
              "duedate": "2024-04-30T00:00:00"
            },
            "relationships": {
              "tool": {
                "data": {
                  "type": "tools",
                  "id": 225455
                }
              },
              "mastermaintenance": {
                "data": {
                  "type": "mastermaintenances",
                  "id": 19
                }
              }
            }
          }
        ]
      }
    }
  }
}
```


This tool only has one maintenance associated with it. You can use the GET method of the MasterMaintenance controller with a specified ID to get the information about this maintenance.

```
GET /MasterMaintenance/19
```

```
{
  "data": {
    "type": "mastermaintenances",
    "id": "19",
    "attributes": {
      "overduedays": 30,
      "warningdays": 7
    },
    "relationships": {
      "mastertool": {
        "data": {
          "type": "mastertools",
          "id": 100123
        }
      },
      "maintenancetype": {
        "data": {
          "type": "maintenancetypes",
          "id": 1
        }
      },
      "toolmaintenances": {
        "data": {
          "type": "toolmaintenancesformastermaintance",
          "id": 19
        }
      }
    }
  }
}
```

You can see that this defines the type of master tool, and the type of maintenance associated with this tool maintenance as well as the overdue days and warning days applied to any instances of this tool maintenance. You could use the MaintenanceTypes controller to get the type of maintenance this is. In this case it is the built-in calibration type.

To update the **datelastmaintenance** and **duedate** attributes, your patch would look like this.

```
PATCH /ToolMaintenancesForTool/225455/19

{
  "data": {
    "type": "toolmaintenanceinstance",
    "id": "225455",
    "attributes": {
      "datelastmaintenance": "2024-04-25T00:00:00",
      "duedate": "2024-05-25T00:00:00"
    }
  }
}
```

Upon issuing this update through the API the maintenance pending status is cleared in both the admin application and on the toolbox.

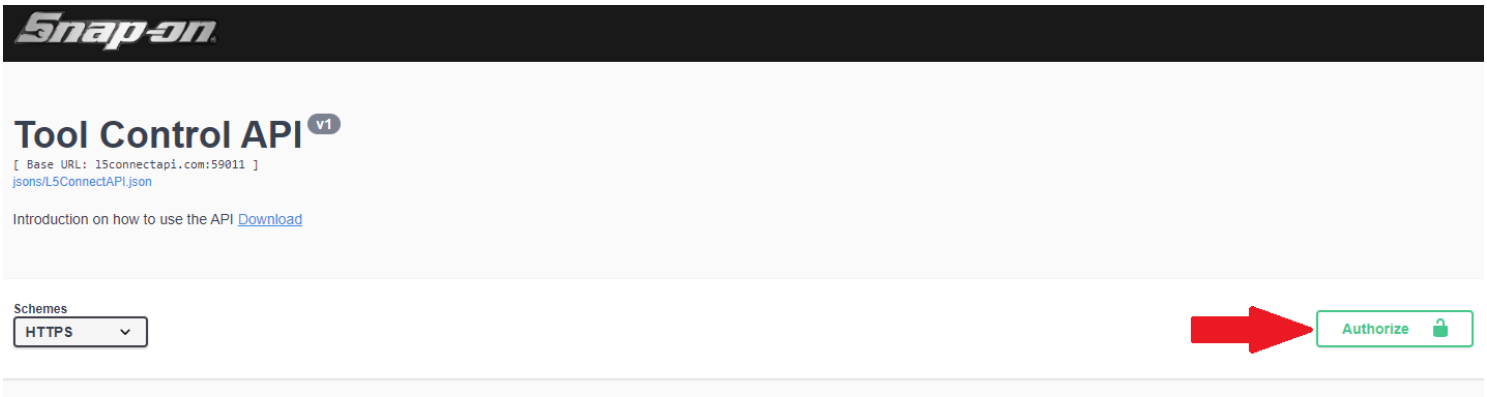
Try it out

Below is a link to a cloud implementation of the API that defines all of the controllers and methods available. You can actually send sample commands to a demo version of the API and see the results.

L5 Connect API

You will need a token to allow you to authenticate your browser session. To obtain a token, contact the Pro Services Team (INDPROSERVICES@snapon.com) and request a token for the L5ConnectApi.com website.

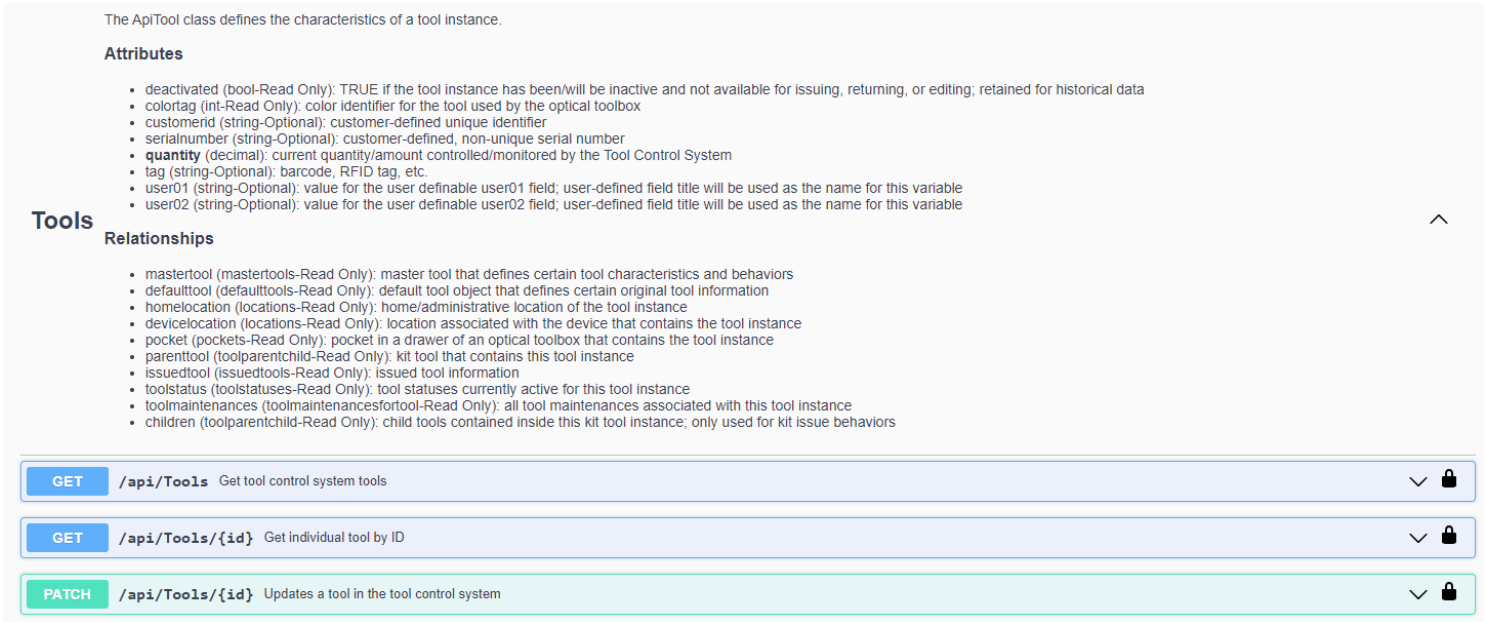
Once you have received your token, click the Authorize button.



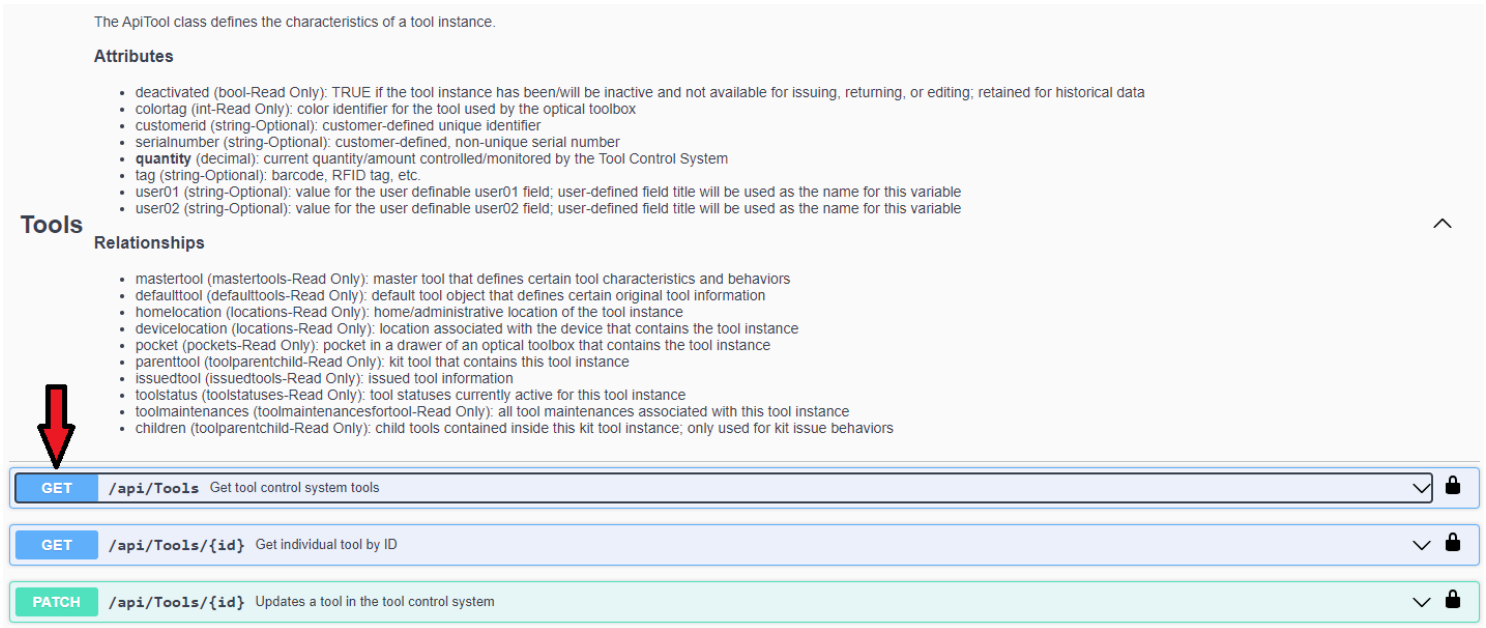
Now paste your token into the Value text box.

Then click the **Authorize** button and then the **Close** button. You are now ready to select a controller method and try it out.

Let's say you wanted a way to get a list of the currently issued tools in the system. The tools controller get method will allow you to get the information you need with the proper filtering. Scroll down to the tools controller.

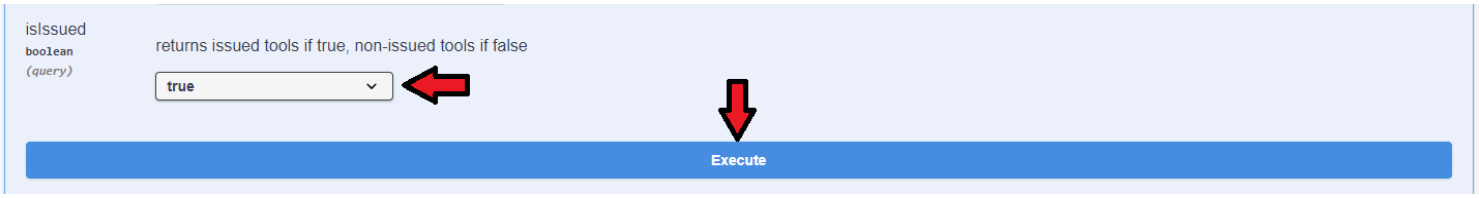


Then click on the **Get /api/Tools** button to expand this get method.

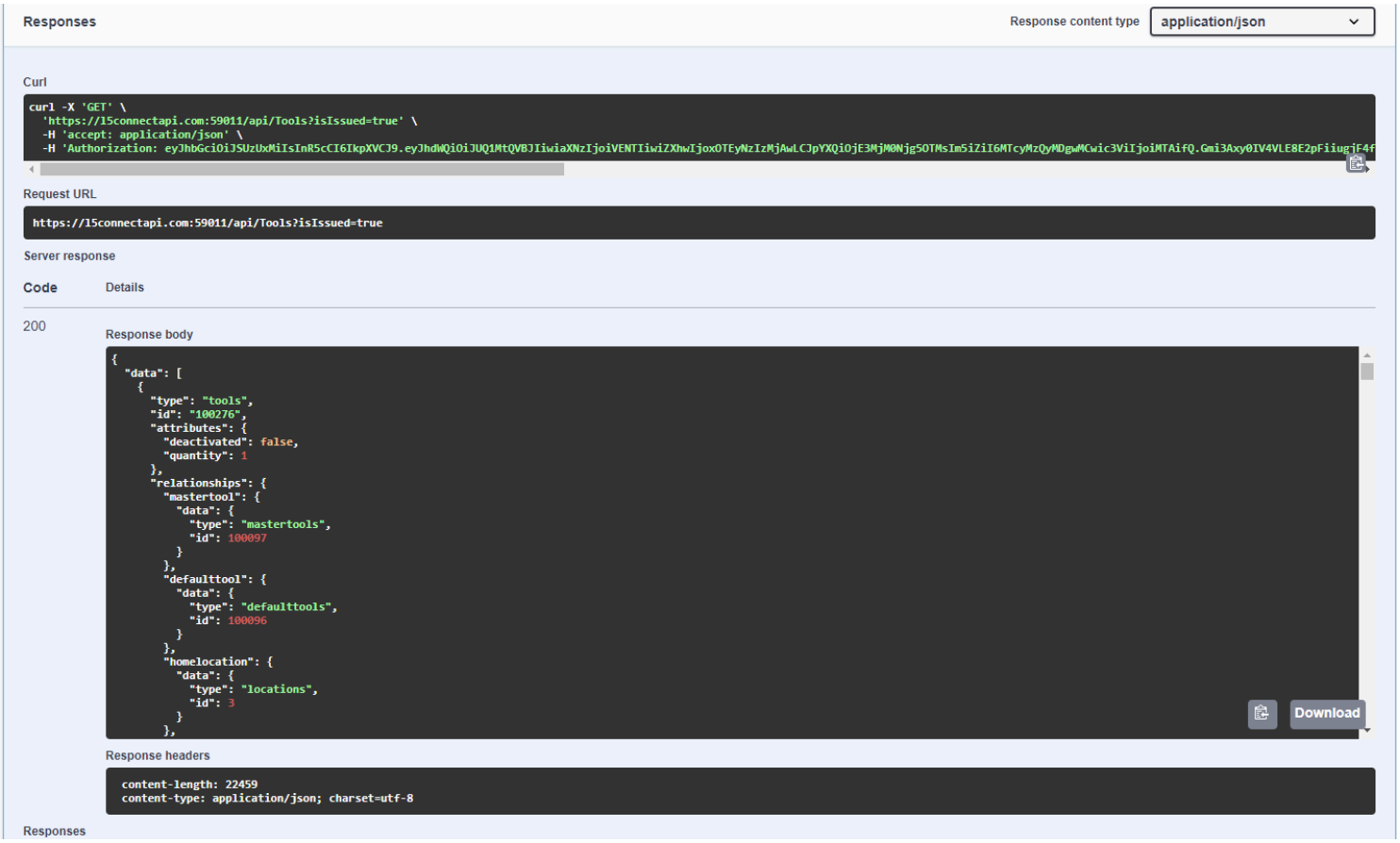


Now click the **Try it out** button to enable the try it out feature.

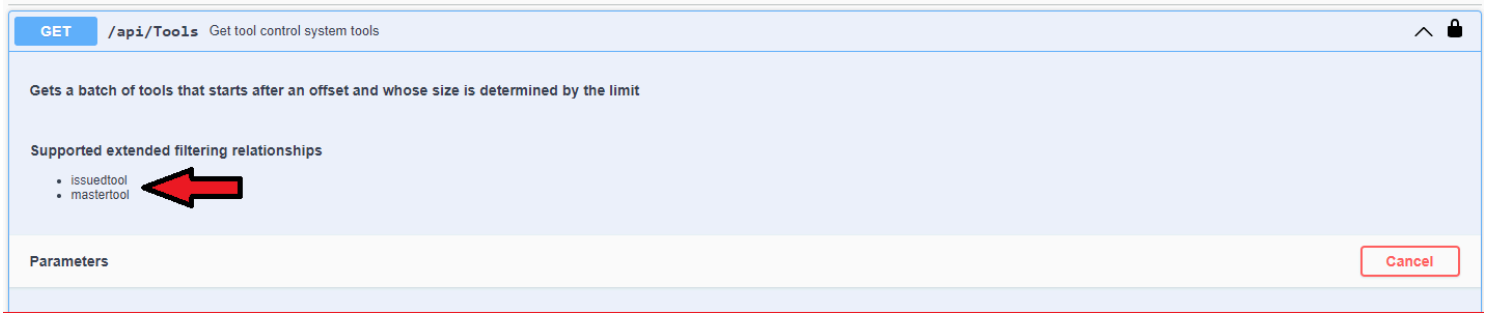
Setting the **isIssued** filter to true will give you a list of all the tools issued in the system. Then click the **Execute** button to try it out.



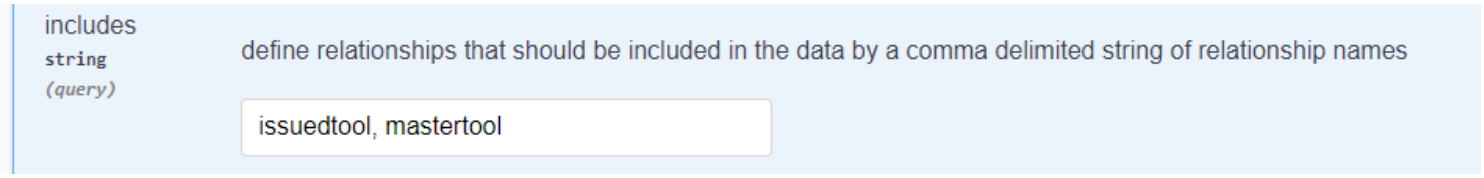
Your response should look something like this.



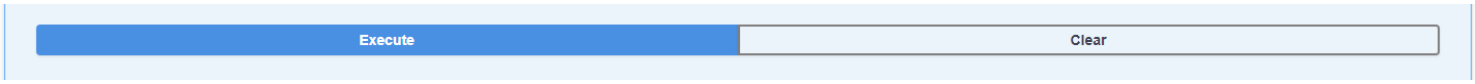
This shows the list of currently issued tools in the system including any attributes the tool has and any relationships it has to other data structures in the system such as its **mastertools** instance or its **issuedtools** instance. Since we are interested in this information it would be nice to get this in this query without having to go to the controllers for those data types and requesting it in separate queries. If we look at the description of the tools controller we can see that it supports extended filtering relationships for both data types.



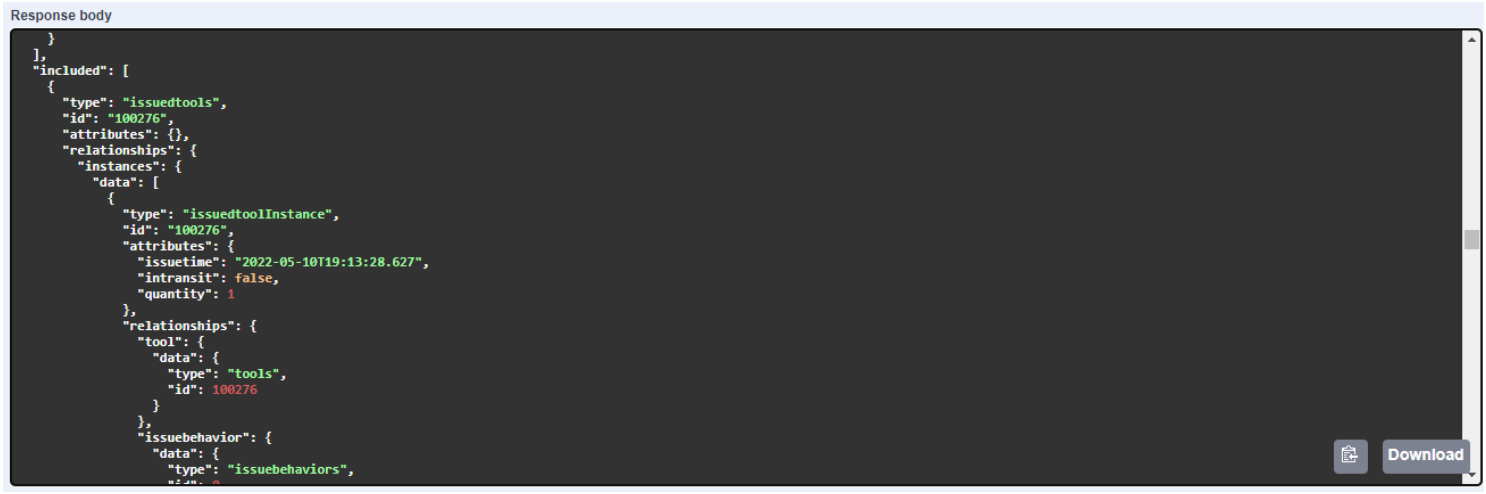
Adding **issuedtool, mastertool** to the includes filter of the request will also include a list of the issued tool relationship data and master tool relationship data for each of the issued tools in the Included part of the response.



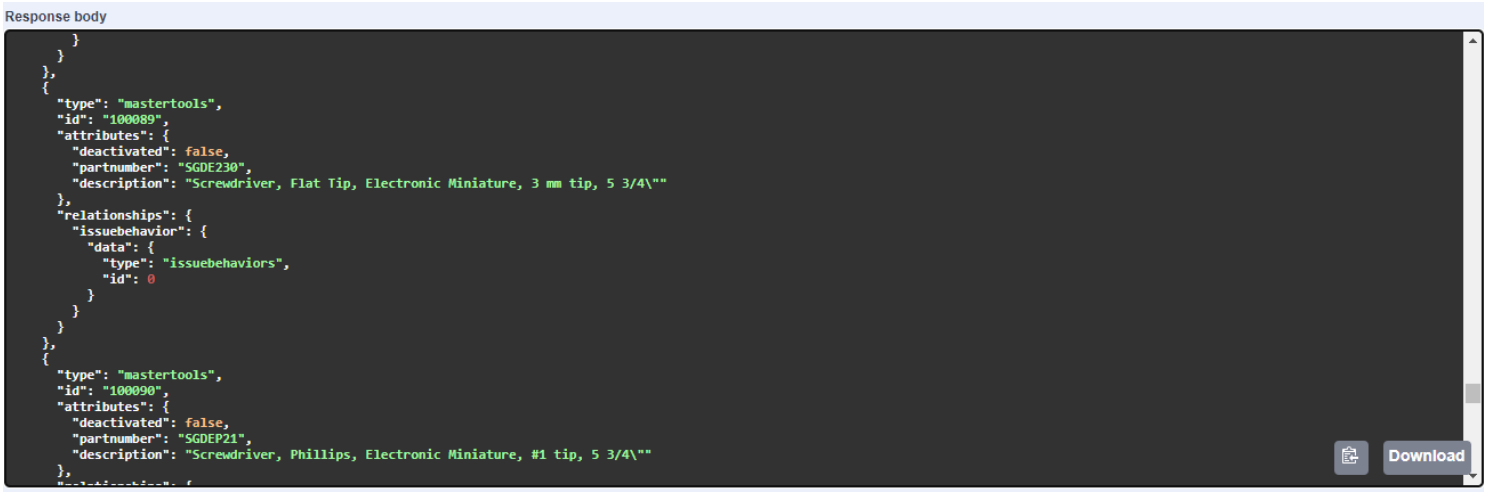
After adding these extended filters click the **Execute** button again.



If you look at the result again it will be the same list of issued **Tools** objects, but if you keep scrolling down to the bottom part of the response you will see an **included** section. This section will have a list of all the issued tool data objects associated with the currently issued tools.



It will also have a list of all the master tool data objects associated with the list of issued tools.



Here is a link of a video of the process.
[Trying out the L5 Connect API]

Conclusion

The L5 Connect API is designed to provide customers with access to the data they need. Snap-on is committed to working with customers as they develop their interface to the L5 Connect API by helping them understand how to use the API and helping them understand where the data they need is in the L5 system.