Developing TeamDynamix Asset Importer Configurations

Contents

Document History

Version	Author	Date	Summary
1	Catherine Stock	2014-04-03	Initial version.
2	Catherine Stock	2014-06-09	Added documentation regarding support for ODBC-based connections.
3	Jamey Stock	2017-10-20	Updated documentation to include ApplicationID.
4	Matt Sayers	2018-05-31	Added documentation for new database command timeout and API request timeout settings.
5	Matt Sayers	2018-11-16	Added capabilities and documentation for name match mappings to support choice fields.

Overview

This document provides a guide to writing the XML configuration file needed to perform an asset import. The import process itself is executed by a command line application running on a client's machine. At a high level, this process is:

1. The command line application loads up the configuration from the specified XML file.
2. The application runs a query against an external database.
3. Each returned row is mapped over to an equivalent API asset instance.
4. The resulting assets are separated into batches and each batch of items is submitted to a REST API method, which will create, update, or ignore assets as appropriate.

Essential Concepts

External Identifiers

The overall importing framework is dependent on having some sort of "external identifier" available as a field on imported items. In the asset management system, this is the "External ID" field. Without an external ID, items submitted to the API will not be accepted. In addition, pre-existing items in TeamDynamix will only be updated if an item submitted to the API matches its external ID. This prevents having runaway assets that would be generated each and every time the import process runs.

For clients that already have many assets in TeamDynamix, this means that the external ID would need to be updated for all of them. However, in 8.6, the existing update functionality of the Excel-based asset import (which will do duplicate checking on the basis of tag), has been extended to support updating external IDs through this process.

The external databases that we will be importing from will have some form of identifier that uniquely identifies an asset in its system. For example, Microsoft's System Center Configuration Manager has a "SMS_Unique_Identifier" column that it uses to store a unique identifier for each system in its database.

Rate Limiting and Batch Sizes

To ensure availability of the TeamDynamix system for all users, the API method used for asset imports is rate-limited. Essentially, this means that only so many calls made to the API method can be performed by a single IP within a certain period of time.

The importer utility is smart enough to detect when it is being rate-limited, and when it encounters such a limit, it will pause until the date/time the rate limit resets, up until a point. If it is instructed to wait more than 10 minutes (which right now is longer than the duration of any rate limit), it will terminate the process. This will work in the vast majority of instances, but individuals running multiple concurrent import processes may experience issues.

In an additional step, the API method also enforces a "batch size", that is, the number of assets that may be provided to the method at any one time. Attempts to submit more than this number of items will result in an error. Note that the batch size is configurable on a per-environment basis, but the rate limit restrictions will always be applied. Both the rate and batch size restrictions are included in the documentation automatically-generated by the web API.

Running the Importer

All of the necessary files have been wrapped up within a standard installer file, which gives you the ability to specify where the files should be located and adds appropriate Start Menu shortcuts.

The importer application itself runs as a command-line application so that it can be easily scheduled as a recurring item and even re-used for multiple import processes (by simply pointing it at a different configuration file every time). The importer application is used like the following:

```
TeamDynamix.ImportUtility.Importer –c "<configuration file>" --add
    --edit
```

In this sample command line, the -c switch indicates the path to the configuration file XML, and the "--add" and "--edit" switches indicate that items should both be created and updated as appropriate. When initially configuring an import process, the following model is recommended:

1. Run the importer without either --add or --edit flags to test that the system will properly query the database, perform mappings, and submit items to the web API without error. Even though items will not be imported, the batch size restriction will still be enforced.
2. Run the importer with the --edit flag to attempt to update pre-existing items, but not create them. A suggested approach is to start out by only having a handful of items in TeamDynamix with an external ID. Once you've validated that the import process will update those assets, you can expand the number of updated assets by setting the external ID on the others.

3. Finally, run the importer with both the --add and --edit flags to perform a full import process. To ensure that duplicate assets will not be added, first ensure that the appropriate external identifier has been set on all relevant assets.

The --add and --edit flags must be explicitly specified. This prevents customers from inadvertently making sweeping changes to their TeamDynamix asset database.

There are more command line options available to the importer, which can be displayed by running the importer with the --help flag or without specifying any configuration file.

It should be noted that any time the importer executes successfully, it will re-save the configuration in the same location to update the last-executed date. This means that any specific formatting and/or comments that may be present in the XML will be removed when it is re-saved.

To preview the results of an import, a separate previewer application has been provided. This application allows you to select an XML configuration file, which will be used to query the external database, attempt any mappings, and highlight any warnings or errors that have been encountered. The resulting preview shows the results of the query side-by-side with the resulting asset.

Sample Import Configuration XML

An example configuration XML file (typically with the extension ".import.xml") looks like the following:

```
<?xml version="1.0" encoding="utf-16"?>
<ImportConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <ObjectType>Asset</ObjectType>
  <ConnectorType>CHANGEME</ConnectorType>
  <ConnectionString>CHANGEME</ConnectionString>
  <DatabaseCommandTimeoutSeconds>30</DatabaseCommandTimeoutSeconds>
  <Query>
      CHANGEME
  </Query>
  <ApiCredentials>
    <ApplicationID>CHANGEME</ApplicationID>
    <BaseApiUrl>CHANGEME</BaseApiUrl>
    <WebServicesBeid>CHANGEME</WebServicesBeid>
    <WebServicesKey>CHANGEME</WebServicesKey>
    <ApiRequestTimeoutSeconds>100</ApiRequestTimeoutSeconds>
  </ApiCredentials>
  <ExternalIdColumnName>CHANGEME</ExternalIdColumnName>
  <BatchSize>1000</BatchSize>
  <Mappings>
    <!-- This is a mapping in which a field is directly-copied from a column returned
by the query. -->
    <SelectedMapping>
      <DestinationFieldId>CHANGEME</DestinationFieldId>
```

```
      <DirectMapColumnName>CHANGEME</DirectMapColumnName>
    </SelectedMapping>
    <!-- This is a mapping in which a series of mapping rules are evaluated and a
default value is applied in the event that none of the rules match. -->
    <SelectedMapping>
      <DestinationFieldId>CHANGEME</DestinationFieldId>
      <MappingRules>
        <MappingRule>
          <RuleType>Equals</RuleType>
          <SourceColumnName>CHANGEME</SourceColumnName>
          <ComparisonValue>CHANGEME</ComparisonValue>
          <MatchValue>CHANGEME</MatchValue>
          <MatchText>CHANGEME</MatchText>
        </MappingRule>
      </MappingRules>
      <DefaultValue>CHANGEME</DefaultValue>
      <DefaultText>CHANGEME</DefaultText>
      <ClearOnNoMapping>false</ClearOnNoMapping>
    </SelectedMapping>
  </Mappings>
  <LastExecutedUtc>0001-01-01T00:00:00</LastExecutedUtc>
  <LastExecutedAsUtc>false</LastExecutedAsUtc>
</ImportConfiguration>
```

Querying the External Database

The **Connector** element indicates which type of database is being accessed. The following connector types are supported:

- **SqlServerConnector** is used to connect to a Microsoft SQL Server database.
- **OdbcConnector** is used to connect to a generic ODBC-accessible database.

The **ConnectionString** element contains the relevant database connection string. This will vary depending on the specified connector.

The **DatabaseCommandTimeoutSeconds** element stores the database command timeout in seconds. The default value is **30**. The default value is used if **DatabaseCommandTimeoutSeconds** is not included in the configuration file or the provided value is less than **30**.

The **Query** element is the actual text of the query that is executed against the database. Note that the mapping rules and logic later on do not currently provide for any data conditioning/normalization, and therefore any necessary normalization should be performed as a part of this query. This query also allows you to use a "@LastExecuted" parameter, the value for which will be specified by the importer utility to allow for the retrieval of partial data sets.

An example of a query might be like the following:

```
<Query>
   Select
      *
   From
      MyAssetTable
   Where
      LastUpdatedDate &gt;= @LastExecuted
</Query>
```

Note on Parameterized ODBC Queries

When an ODBC connection (that is, a configured Connector of "**OdbcConnector**") is being employed, named parameters are not supported. When accessing databases through ODBC, the question mark placeholder ("?") must be used in place of the named @LastExecuted parameter. This parameter is only passed once to the provided query, and therefore only the first placeholder will be provided with a value.

Note that because this is in XML, the ">" symbol has been escaped as ">". Other entities may need to be similarly-escaped.

For more information about how a sample SCCM query against a SQL Server database (SqlServerConnector) might execute, see the included appendix.

Configuring API Credentials

The **ApiCredentials** element stores the information necessary to connect to and authenticate against the web API, and acts as a container for the other elements identified within this section.

The **ApplicationID** element contains the ID of the application into which the items will be imported. If this is omitted, then the ID of the default Assets/CIs application for your organization will be used.

The **BaseApiUrl** element indicates the base URL, such as https://app.teamdynamix.com/TDWebApi/, for the REST API.

The **WebServicesBeid** and **WebServicesKey** elements are GUIDs that are used to authenticate against the web API using the "API User" account for the BE. This information can be retrieved from the detail page for a BE in TDAdmin.

The **ApiRequestTimeoutSeconds** element stores the request timeout in seconds for all API calls this utility needs to make. The default value is **100**. The default value is used if **ApiRequestTimeoutSeconds** is not included in the configuration file or the provided value is less than **100**.

Alternatively, authentication can be performed using **ApiUsername** and **ApiPasswordEncrypted** elements. However, because getting the encrypted version of the password will most likely involve developer intervention, this method is not recommended for typical use.

Finally, an optional **ProxyUrl** element allows you to define a proxy URL through which all API traffic will be routed. This allows you to use an application such as Fiddler to view the raw content and response of each API call.

Other Important Settings

External Identifier

The **ExternalIdColumnName** element indicates the column returned by the query that is used to determine the external ID of any asset. This value is directly-copied from the column to the asset's external ID.

Batch Size

The **BatchSize** element indicates the maximum number of assets that will be included in each API call. As previously-mentioned, this can go no higher than the limit configured for the environment, but smaller batches can be specified as necessary.

Last-Execution Date

The **LastExecutedUtc** element stores the last date/time the import process was successfully executed. This will be automatically updated by the importer utility, but can be edited as necessary.

The **LastExecutedAsUtc** element indicates if the database expects this value to be in UTC. When this is **false**, the importer utility will convert the last-executed date to the *machine's* local time (note that this may differ from the *database's* local time) before providing it to the query.

Configuring Mappings

The **SelectedMappings** element contains all of the mappings for individual asset fields. This contains any number of **SelectedMapping** elements, although you will at least need mappings for the following required fields:

- SerialNumber or Name (only one must be provided)
- StatusID

Note that the external ID of an asset is handled automatically by the **ExternalIdColumnName** configuration mentioned earlier and therefore cannot be mapped through this process.

Specifying Fields

All selected mappings must have a **DestinationFieldId** element provided to indicate which field will be updated by the mapping. The set of standard fields is predefined, and so a listing of available standard asset fields is available in a separate appendix.

If an asset field does not have a mapping provided for it, the web API will ignore it when it attempts to update pre-existing assets. This ensures that values will not be inadvertently cleared through an update process.

Custom attributes will vary on a per-organization basis, but can still be set during the import process. To set the value of a custom attribute, use a **DestinationFieldId** value in the following format:

```
<DestinationFieldId>CustomAttribute-<attribute ID></DestinationFieldId>
```

For example, you may want to set a custom attribute such as "IP Address" (with an ID of 1000) and directly-copy it from the value of an "IPAddress" column returned by the database query. This would entail a mapping like the following:

```
<SelectedMapping>
    <DestinationFieldId>CustomAttribute-1000</DestinationFieldId>
    <DirectMapColumnName>IPAddress</DirectMapColumnName>
</SelectedMapping>
```

With choice fields, such as **SupplierID**, you may want to perform a name match lookup with the database value to find the TeamDynamix choice ID to send in the import. For example, you might have a Manufacturer name of “Dell” in the database and you want to map that to your TeamDynamix Supplier choice of “Dell” (with an ID of 100). To do this, you would utilize a name match mapping to dynamically find the TeamDynamix Supplier ID for the import like the following:

```
<SelectedMapping>
    <DestinationFieldId>SupplierID</DestinationFieldId>
    <NameMatchColumnName>ManufacturerName</NameMatchColumnName>
</SelectedMapping>
```

Similarly, to name match map a custom attribute which is a choice-based custom attribute (where the custom attribute ID is 1100), you would do the following:

```
<SelectedMapping>
    <DestinationFieldId>CustomAttribute-1100</DestinationFieldId>
    <NameMatchColumnName>OperatingSystem</NameMatchColumnName>
</SelectedMapping>
```

In other instances, you may wish to use a series of mapping rules or a default value for a custom attribute field. When doing so for custom attributes with choices, you will need to use the ID values of the custom attribute choices you wish to use.

Furthermore, when a custom attribute field that supports multiple values (such as a multiselect or checkbox list) is being mapped, you can use a comma-separated list of choice IDs to represent multiple choice selections. For example, imagine a "Roles" custom attribute (with an ID of 2000) that has multiple choices for indicating how a particular asset is being used. Using a default role set of "Workstation" and "General Purpose" would look like the following:

```
<SelectedMapping>
   <DestinationFieldId>CustomAttribute-2000</DestinationFieldId>
   <DefaultValue>3000,3050</DefaultValue>
   <DefaultText>Workstation,General Purpose</DefaultText>
</SelectedMapping>
```

Directly-Copying Columns

The simplest way to import values from a database query is to directly copy a value from a column. This will typically be the case when there are plain "value" (and not "choice") fields you wish to set, such as the serial number and asset tag. In this case, the SelectedMapping element should contain a **DirectMapColumnName** element with the name of the appropriate column, such as in the example above.

Name Matching Columns

Overview

For choice-based standard and custom attribute fields, the most common way to import values from the query is to utilize name match mapping. This mapping type takes the column value from the database query and looks up the actual TeamDynamix choice ID(s) from the TeamDynamix API behind the scenes.

It should be noted that all name match query values should be strings, or converted to strings, for name matching to work properly. All name matching is performed in a **case-insensitive** manner, meaning that casing differences *will not* impact the matching logic.

To use a name match mapping, the SelectedMapping element should contain a **NameMatchColumnName** element with the name of the appropriate column, such as in the examples above.

Matching Multiple Choices

For fields that support multiple choice values, each choice name should be separated by the **| (pipe)** character in the query value used for matching.

Default Value Support

Default values and text are supported for name match mappings, similar to the examples above, by including **DefaultValue** and **DefaultText** elements. This works exactly as default values and text work for direct match mappings.

When No Value is Found

If a name match mapping is specified but no value is found in the TeamDynamix choice data from the API, the record(s) in question will simply be treated as if that mapping were not specified in the configuration at all. At this point, if there is a default value specified and it is valid, that value will be used if a new item needs to be created.

If there is a default value specified but the item already exists server-side, the destination field *will not* be updated as default values are only for creation. In this case the destination field value will not be changed server-side.

If there is no default value specified, a warning or error message will be written to the log file depending upon if the destination field is required or not. In this case the destination field value will not be changed server-side.

Name Match Support Fields and Mapping Logic

The following table describes the TeamDynamix destination fields supported for name match mappings and how those mappings are performed.

Destination Field Name	Matches On	Ex. Query Value(s)	Ex. Result Value(s)
SupplierID	Supplier Name	**Dell**	100
ProductModelID	Supplier Name|Model Name	**Dell|Inspiron 3670**	150
StatusID	Status Name	**In Use**	10
LocationAndRoomID	Location Name or External ID|Room Name or External ID	**Archer Hall|Room 123** OR **AH176|123**	50,125
OwningDepartmentID	Acct/Dept Code or Name	**Student** OR **STU001**	75
OwningCustomerID*	1. Username 2. Auth Username 3. Alternate ID 4. Organizational ID	**john.doe@school.edu** OR **john.doe** OR **jdoe11223** OR **11223**	3d89a141-4c38-4af0-9810-63b3532a6942
RequestingDepartmentID	Acct/Dept Code or Name	**Student** OR **STU001**	75
RequestingCustomerID*	1. Username 2. Auth Username 3. Alternate ID 4. Organizational ID	**john.doe@school.edu** OR **john.doe** OR **jdoe11223** OR **11223**	3d89a141-4c38-4af0-9810-63b3532a6942
ParentID (Parent Asset)	Asset External ID	**88967**	5000
MaintenanceScheduleID	Maintenance Schedule Name	**All Fridays Blackout**	1275
Single-Choice Custom Attributes (Ex: OS Name)	Choice Name	**Windows**	1252
Multiple-Choice Custom Attributes (Ex: Affected Versions)	Choice Name|Choice Name|Choice Name	**7|8|8.1**	10,20,30

**** Person choices are matched in a waterfall fallout style on the fields and order specified in the Matches On column.***

Mapping Rules

When you wish to use more complex logic, particularly in the case of "choice" fields such as supplier, product model, or location, the system allows you to configure one or more mapping rules. Each mapping rule will be evaluated in the order they are listed, and the system will stop evaluating mapping rules once it finds a match.

Each individual **MappingRule** element contains the following sub-elements:

- **RuleType** - the type of rule to evaluate.
- **SourceColumnName** - the name of the column being evaluated.
- **ComparisonValue** - the value against which the source column is compared.
- **MatchValue** - the value to use when the rule evaluates successfully.
- **MatchText** - the text representation of the match value. This element is not used by the system, but can be used by you to help describe what the match value actually represents. For example, you could be evaluating a mapping rule for an asset's supplier and wish to have a rule for a "VMware" supplier with an ID of 47. In such an instance, you could use a MatchValue of 47 and a MatchText of "VMware" to act as a reminder that an ID of 47 corresponds with that supplier.

In addition, each SelectedMapping element can contain a **ClearOnNoMapping** sub-element to indicate that a value should be cleared out when none of the mapping rules successfully match. Note that this will not be valid for required fields such as supplier and product model. For such fields, you will also want to provide a default fallback value to ensure that assets can be successfully created/updated.

Note that when the value of a particular column evaluates to null when it is referenced in a mapping rule, the rule always fails.

Mapping Rule Types

There are many types of different mapping rules that can be evaluated through this process. Note that some mapping rules require that the value(s) being compared fulfill some basic data type restrictions.

The set of available rule types is summarized below:

Rule Type	Description	Column Type Restrictions
Equals	Checks if the two values are equal.	None
EqualsNot	Checks if the two values are not equal.	None
Contains	Checks if the column contains the string represented by the comparison value.	Text
ContainsNot	Checks if the column does not contain the string	Text

	represented by the comparison value.	
RegularExpression	Checks if the column matches the regular expression represented by the comparison value.	Text
RegularExpressionNot	Checks if the column does not match the regular expression represented by the comparison value.	Text
GreaterThan	Checks if the column is greater than the comparison value.	Numeric or date/time
GreaterThanEqual	Checks if the column is greater than or equal to the comparison value.	Numeric or date/time
LessThan	Checks if the column is less than the comparison value.	Numeric or date/time
LessThanEqual	Checks if the column is less than or equal to the comparison value.	Numeric or date/time

When equality or inequality checks are being performed, the column value and comparison value must have equivalent data types. Any string-comparisons will be case-sensitive.

Type mismatches will result in the mapping rule failing evaluation.

Default Values

In some instances, you may want to provide a fall-back or standard value for an asset field. This is particularly appropriate for required "choice" fields such as supplier, product model, or status.

Configuration of a default value is specified through a **DefaultValue** sub-element within your SelectedMapping element. To aid in identifying what a default value represents, you can include a **DefaultText** element as a text representation of the value. This parallels the MatchValue/MatchText elements available for individual mapping rules, and thus the importer will ignore what has been specified for this element.

In some instances, you may just simply wish to provide a default value without attempting any mapping logic whatsoever. The use of a default value does not require the use of any mapping rules. This sample mapping always sets the same status for any assets:

```
<SelectedMapping>
   <DestinationFieldId>StatusID</DestinationFieldId>
   <DefaultValue>219</DefaultValue>
   <DefaultText>Detected by Scanner</DefaultText>
</SelectedMapping>
```

Note that when a pre-existing asset is being updated through the import process, the API method will ignore default values and instead prefer the existing field value. This means that the import process can never "reset" a field on a pre-existing asset. This is somewhat of a limitation in functionality, but is an acceptable trade-off to prevent clearing values that were later manually updated.

For example, if a status of "Detected by Scanner" is used for an asset when it is initially imported, and a technician later goes in and updates the status to "In Use - Computing Lab" or "In Use - Personal Machine", any subsequent updates will preserve what has been manually entered for the status.

For required fields, it is advisable to configure sentinel values of "Unknown". This can be used in conjunction with the mapping logic, so that you can have a base set of mapping rules to cover the majority of common values and fall back to that sentinel value for any edge cases.

This sample product model mapping attempts to cover some basic matching against the "ComputerSystem-Model" field to try and identify VMware virtual machines and Cisco Catalyst 6500 devices, but will fall back to a standard "Unknown" field in the event that the model is not either of these values:

```
<SelectedMapping>
   <DestinationFieldId>ProductModelID</DestinationFieldId>

   <MappingRules>

      <MappingRule>
         <RuleType>Contains</RuleType>
         <SourceColumnName>ComputerSystem-Model</SourceColumnName>
         <ComparisonValue>VMware Virtual</ComparisonValue>
         <MatchText>VMware - Virtual Machine</MatchText>
         <MatchValue>79</MatchValue>
      </MappingRule>

      <MappingRule>
         <RuleType>Equals</RuleType>
         <SourceColumnName>ComputerSystem-Model</SourceColumnName>
         <ComparisonValue>Catalyst 6500</ComparisonValue>
         <MatchText>Cisco - Catalyst 6500</MatchText>
         <MatchValue>81</MatchValue>
      </MappingRule>

   </MappingRules>

   <DefaultValue>80</DefaultValue>
   <DefaultText>Unknown</DefaultText>

</SelectedMapping>
```

Appendix A: Asset Fields

For the most part (with the exception of the "LocationAndRoomID" field), these fields are the same as those available on the API asset object, and so descriptions for them can be viewed through the standard web API documentation. When writing an import configuration, you do not need to provide mappings for every single field, although all required fields must have a mapping.

The following standard asset fields are pre-defined:

Field Name	Req'd?	Format of MatchValue/DefaultValue	Example
SerialNumber	***Yes****	String	DG749823K
Name	***Yes****	String	Conference Room Table
Tag	No	String	TG-0001
SupplierID	No	Vendor ID	444
ProductModelID	No	Product model ID	333
StatusID	***Yes***	Asset status ID	21
LocationAndRoomID	No	**<location ID>,<room ID>**	20,30
OwningDepartmentID	No	Account/department ID	2048
OwningCustomerID	No	Person UID	3d89a141-4c38-4af0-9810-63b3532a6942
RequestingDepartmentID	No	Account/department ID	1024
RequestingCustomerID	No	Person UID	4d0f8e0a-f168-44b1-b708-0e03d05cdef1
AcquisitionDate	No	**YYYY-MM-DDTHH:MM:SS**	2012-06-05T00:00:00
ExpectedReplacementDate	No	**YYYY-MM-DDTHH:MM:SS**	2014-06-05T00:00:00
PurchaseCost	No	Number (no currency symbol)	100.00
ParentID	No	Asset ID	2095
MaintenanceScheduleID	No	Maintenance schedule ID	1988

**** This is only required if the other value has not been provided. Assets must have Name or Serial #***

A value of "0" indicates when no account/department is specified. Similarly, a value of "0" must be used when no parent asset or maintenance schedule is specified. An empty UID ("00000000-0000-0000-0000-000000000000") must be used when no customer is indicated.

The LocationAndRoomID exists as a composite field. Values for this field should be formatted in "***<Location ID>,<room ID>***" format, and the presence of "0" for an ID indicates that a location or room is not being specified.

Appendix B: Sample SCCM Query

Microsoft's System Center Configuration Manager has a standard, well-documented schema which is heavily relied-on in the query described here. This query selects from the standard "v_R_System" view, which represents all System-identified resources within SCCM. This query also assumes that the specified connector is for a SQL Server database (SqlServerConnector).

This query takes the following steps:

1. Top-level information about each system is stored in a temporary table. This includes associated information about the enclosure, operating system, and the most-specific name for the system in the organizational unit.
2. Network adapter information for all retrieved systems is stored in a separate temporary table. The "ResourceID" column indicates which systems correspond with which network adapter. An additional "ResourceIndex" column is stored to help identify ordering when multiple network adapters are associated with a single system.
3. The contents of the first temporary table are returned, along with joining to the temporary table with network adapter information. Network adapters with a ResourceIndex of 1 or 2 are returned back as separate column sets (with "NetworkAdapter0-" and "NetworkAdapter1-" prefixes) to flatten out the result of the query.

In addition, this query performs last-updated filtering on the basis of the "LastHWScan" value available in the associated "v_GS_WORKSTATION_STATUS" record. The exact value used to perform this filtering may differ on a per-organization basis.

```
If OBJECT_ID('tempdb..#ComputerSystems') Is Not Null Begin
  Drop Table #ComputerSystems
End

If OBJECT_ID('tempdb..#NetworkAdapters') Is Not Null Begin
  Drop Table #NetworkAdapters
End

Select
  -- Top-level attributes on the system
  rs.ResourceID as 'ResourceID',

  rs.Active0 as 'Active',
  rs.AD_Domain_Name0 as 'AD_Domain_Name',
  rs.AD_Site_Name0 as 'AD_Site_Name',
  sysoun.System_OU_Name0 as 'OU_Name',
  rs.Creation_Date0 as 'Creation_Date',
  rs.Decommissioned0 as 'Decommissioned',
  rs.Hardware_ID0 as 'Hardware_ID',
  rs.User_Domain0 as 'LastLogon_UserDomain',
  rs.User_Name0 as 'LastLogon_UserName',
  rs.Name0 as 'Name',
  rs.Netbios_Name0 as 'Netbios_Name',
  rs.Obsolete0 as 'Obsolete',
```

```
  rs.Previous_SMS_UUID0 as 'Previous_SMS_UUID',
  rs.Primary_Group_ID0 as 'Primary_Group_ID',
  rs.Resource_Domain_OR_Workgr0 as 'Resource_DomainOrWorkgroup',
  rs.SMBIOS_GUID0 as 'SMBIOS_GUID',
  rs.SMS_Unique_Identifier0 as 'SMS_Unique_Identifier',
  rs.SMS_UUID_Change_Date0 as 'SMS_UUID_Change_Date',
  rs.Community_Name0 as 'SNMP_Community_Name',
  rs.User_Account_Control0 as 'User_Account_Control',

  -- System attributes
  syst.Domain0 as 'System-Domain',
  syst.Name0 as 'System-Name',
  syst.SystemRole0 as 'System-System_Role',

  -- Enclosure attributes
  encl.Manufacturer0 as 'Enclosure-Manufacturer',
  encl.SerialNumber0 as 'Enclosure-SerialNumber',
  encl.SMBIOSAssetTag0 as 'Enclosure-SMBIOS_Asset_Tag',
  encl.Tag0 as 'Enclosure-Tag',

  -- Computer system attributes
  compsys.Manufacturer0 as 'ComputerSystem-Manufacturer',
  compsys.Model0 as 'ComputerSystem-Model',
  compsys.NumberOfProcessors0 as 'ComputerSystem-NumberOfProcessors',
  compsys.Roles0 as 'ComputerSystem-Roles',
  compsys.Status0 as 'ComputerSystem-Status',
  compsys.UserName0 as 'ComputerSystem-UserName',

  -- OS Attributes
  rs.Operating_System_Name_and0 as 'OS-NameAndVersion',
  opsys.Caption0 as 'OS-Caption',
  opsys.CSDVersion0 as 'OS-CSDVersion',
  opsys.Version0 as 'OS-Version',

  -- Workstation status attributes
  wkst.LastHWScan as 'Workstation-LastHardwareScan',
  wkst.LastReportVersion as 'Workstation-LastReportVersion'
Into
  #ComputerSystems
From
  dbo.v_R_System rs
  Left Outer Join dbo.v_GS_SYSTEM syst On rs.ResourceID = syst.ResourceID
  Left Outer Join dbo.v_GS_WORKSTATION_STATUS wkst On rs.ResourceID = syst.ResourceID
  Left Outer Join dbo.v_GS_COMPUTER_SYSTEM compsys On rs.ResourceID = compsys.ResourceID
  Left Outer Join dbo.v_GS_OPERATING_SYSTEM opsys On rs.ResourceID = opsys.ResourceID
  Left Outer Join dbo.v_GS_SYSTEM_ENCLOSURE encl On rs.ResourceID = encl.ResourceID
  Outer Apply (
    /* Get the most-specific OU name, hence the length ordering involved */
    Select
      Top(1)
      oun.System_OU_Name0
    From
      dbo.v_RA_System_SystemOUName oun
```

```
    Where
      oun.ResourceID = rs.ResourceID
    Order By
      LEN(oun.System_OU_Name0)
  ) sysoun
Where
  /* Perform last-modified filtering */
  IsNull(wkst.LastHWScan, GETDATE()) >= @LastExecuted

/* Get back network adapters for each system above that was retrieved */
Select
  -- Network adapter options
  na.MACAddress0 as 'MACAddress',
  na.Name0 as 'Name',
  na.ServiceName0 as 'ServiceName',

  -- Configuration options
  nac.DefaultIPGateway0 as 'DefaultIPGateway',
  nac.DHCPEnabled0 as 'DHCPEnabled',
  nac.DHCPServer0 as 'DHCPServer',
  nac.DNSDomain0 as 'DNSDomain',
  nac.DNSHostName0 as 'DNSHostName',
  nac.IPAddress0 as 'IPAddress',
  nac.IPSubnet0 as 'IPSubnet',

  -- Thise is used to link back to the associated network device, and also flatten
  -- out the order to get some hierarchy
  na.ResourceID,
  ROW_NUMBER() OVER (PARTITION BY na.ResourceID ORDER BY nac.Index0) as ResourceIndex
Into
  #NetworkAdapters
From
  dbo.v_GS_NETWORK_ADAPTER na
  Inner Join dbo.v_GS_NETWORK_ADAPTER_CONFIGUR nac On na.ResourceID = nac.ResourceID
    And na.DeviceID0 = nac.Index0
  Inner Join #ComputerSystems cs On na.ResourceID = cs.[ResourceID]
Where
  /* This is a fairly-standard set of filtering parameters used to weed out unnecessary
    items such as loopback adapters. */
  na.AdapterType0 = 'Ethernet 802.3'
  And na.Description0 Not Like '%Miniport%'
  And nac.IPEnabled0 = 1
  And na.MACAddress0 Is Not Null

/* Now select back all of the base computer information and
  a flattened-out version of the network adapter information */
Select
  cs.*,

  -- Network adapter/configuration information
  na_1.[Name]as'NetworkAdapter0-Name',
  na_1.[MACAddress]as'NetworkAdapter0-MACAddress',
  na_1.[ServiceName]as'NetworkAdapter0-ServiceName',
```

```
  na_1.[DefaultIPGateway]as'NetworkAdapter0-DefaultIPGateway',
  na_1.[DHCPEnabled]as'NetworkAdapter0-DHCPEnabled',
  na_1.[DHCPServer]as'NetworkAdapter0-DHCPServer',
  na_1.[DNSDomain]as'NetworkAdapter0-DNSDomain',
  na_1.[DNSHostName]as'NetworkAdapter0-DNSHostName',
  na_1.[IPAddress]as'NetworkAdapter0-IPAddress',
  na_1.[IPSubnet]as'NetworkAdapter0-IPSubnet',

  na_2.[Name]as'NetworkAdapter1-Name',
  na_2.[MACAddress]as'NetworkAdapter1-MACAddress',
  na_2.[ServiceName]as'NetworkAdapter1-ServiceName',
  na_2.[DefaultIPGateway]as'NetworkAdapter1-DefaultIPGateway',
  na_2.[DHCPEnabled]as'NetworkAdapter1-DHCPEnabled',
  na_2.[DHCPServer]as'NetworkAdapter1-DHCPServer',
  na_2.[DNSDomain]as'NetworkAdapter1-DNSDomain',
  na_2.[DNSHostName]as'NetworkAdapter1-DNSHostName',
  na_2.[IPAddress]as'NetworkAdapter1-IPAddress',
  na_2.[IPSubnet]as'NetworkAdapter1-IPSubnet'
From
  #ComputerSystems cs
  /* Filter on ResourceIndex here to flatten out the hierarchy and prevent multiple rows from
    being returned. /*
  Left Outer Join #NetworkAdapters na_1 On cs.ResourceID = na_1.ResourceID
    And na_1.ResourceIndex = 1
  Left Outer Join #NetworkAdapters na_2 On cs.ResourceID = na_2.ResourceID
    And na_2.ResourceIndex = 2
```