Unveiling TADDM location tagging

Document version 1.0

July 15, 2013

Morten Moeller (moellerm@us.ibm.com)

Introduction

IBM® Tivoli® Application Dependency Discovery Manager (TADDM) v7.2.1 FP3 provides a feature for automated location tagging. The accompanying TADDM documentation provides an introduction to the use of this feature; however it does not cover the side effects related to the use of access control lists that are introduced by enabling the location tagging feature.

This paper describes how to use the location tagging feature, and how the use of credentials stored in the TADDM access control list are affected when the location tagging feature is enabled. In addition best practice recommendations on how to maintain the location attribute of the CIs stored in the TADDM database – with or without use of the location tagging feature – is provided. In addition, this paper also provide guidance on how to visualize the location information in the TADDM GUI, and how to consume the location information in BIRT and Cognos reports

In the appendix of this paper you will also find a custom server extension script that mimics the functionality for tagging ComputerSystems with a location, based on the ComputerSystems relationship to scopes and the association between anchor hosts and scopes. Using this extension, you can get the benefits of automatically populating the locationTag attribute for your ComputerSystem resources, without having to modify your credentials.

Table of Contents

1 Overview

Before digging into the details of how to configure and use TADDM location tagging let's take a quick look at what the location tagging feature does, and which benefits it provides.

1.1 What is location tagging?

Location tagging is a term used to describe a feature in TADDM that associates each discovered component with a specific location. For service providers that use a shared TADDM instance to support multiple organizations this feature is especially interesting because the location information can be used by TADDM or other tools that leverage the TADDM database for example to

- Implement data isolation in shared environments
- Control access to component details through access collections.
- Produce location (or customer) specific reports.
- To automatically enrich, and route events and tickets to a customer specific support organization.

By associating each discovered component with a specific location, this information can be used by most service management tools, including IBM SmartCloud Control Desk, as a filtering criteria to enable **data isolation** and **access control**. This is one of the primary reasons for using the TADDM built-in location tagging feature so that the locationTag attribute for all of the components identified by the discovery is automatically populated.

In addition, the location tagging feature provides **credential isolation** between different parts of your discovered infrastructure. The feature ensures that credentials that belong to one part of the infrastructure are not exposed to sensors and template extension scripts that operate in the context of another part of the infrastructure.

In essence, location tagging introduces a logical grouping, based on the location, which is used during discovery to use and expose only credentials that are associated with the specific location context in which the discovery executes. The location context is determined during discovery based on the anchor host that handles the discovery. Typically the location tagging feature is beneficial in shared TADDM environments that support multiple customers or multiple isolated environments, where you want to ensure that credential spoofing cannot take place.

Background:

During discovery, sensors and extension scripts can query the TADDM database to obtain credentials that support specific component types, for example ComputerSystems, or AppServers. If location tagging is not enabled, all credentials that support the requested resource type and IP Address (based on optional scope restrictions) will be reveiled to the script or sensor. In a shared environment, this imlementation provides a security exposure because TADDM cannot control which types of credentials are queried for which IP Addresses by the template extension scripts.

When enabling location tagging, only the credentials that belong to the location that is associated with the anchor host responsible for the discovery are exposed, and thus an extension script cannot access credentials that belong to another location.

When you decide to enable location tagging, you take on additional administration tasks related to maintenance of the credentials defined to TADDM. When location tagging is enabled, only credentials that are associated with the discovery context - determined by the anchor host location - can be used during discovery. For that reason, you can no longer use credentials that are shared between locations and you may have to replicate your shared credentials so that they are represented in all the locations you wish to support.

As an added benefit, when location tagging is enabled, the locationTag attribute is populated automatically for all the components that are discovered. The value that is used to populate the locationTag attribute is determined by the discovery context, and thusly, by the location that is assigned to the anchor host. In TADDM, some components, such as external relationships and aggregation, and inherited components are created by background topology agents based on analysis of the discovered components. The locationTag is also assigned to these background-created components under certain conditions. The locationTag attribute of these computed components is populated only if:

- A one-to-one relationship (such as Dependency or NetworkConnection) includes a value for the locationTag attribute if the location is the same for both connected components.
- An aggregation component (such as a cluster) includes a value for the locationTag attribute if the location is the same for all aggregated objects.
- Simple components include a value for the locationTag attribute from the component it is based on.

In all other cases, components that are created by the background topology agents do not include a location tag value.

If you want to populate the locationTag without enabling the location tagging feature you can easily deploy your own template extensions to accomplish this task. Using extension scripts also provides additional freedom to apply your own logic when assigning values to the locationTag attribute. However, using extension scripts also provides an administrative overhead. In addition, extension scripts do not apply to level 1 discovery, and is (in TADDM V7.2.1) limited to resources for which you can associate templates (ComputerSystems and Custom Application Servers). This restriction may be lifted in upcoming releases.

Either way, populating the *locationTag* attribute of the CIs that are stored in the TADDM database enables you to keep track of the physical location of your IT resources.

If you have enabled location tagging, the locationTag attribute can also be populated while importing components from a discovery library book.

Using location tagging, a location can be assigned to a resource in one of two ways:

Statically	By associating each discovery server and anchor host with a specific location. Depending on the scopes associated with each anchor host, the location of the resources discovered by the anchor is set to the pre-defined value for the anchor host.
Dynamically	By providing a specific location value to a command-line discovery or a bulkload command.

If location tagging is not enabled, a value can be assigned to the locationTag attribute during discovery:

At discovery	By associating a template extension to your templates and apply your own logic to determine the location. This can be based on hostnames, fqdn, IP address or IP domain, MAC address, switch connection, or any other discovered attribute or relationship you want to use.

Independently of whether location tagging is enabled or not, a value can be assigned to the locationTag attribute in one of these ways:

Programmatically	By creating a program or script that uses any of the available APIs which allow you to update information in the TADDM database. Currently the Java, REST, and SOAP APIs can be used. As an example, you can check out the rest_in_ease utility available from the IBM Integrated Service Management Library which can be used to update any attribute from a command line. You can find the utility at https://www.ibm.com/software/brandcatalog/ismlibrary/details?catalog.label=1TW10TA35.
Manually	By using the TADDM command-line API to export the relevant definitions, manually update the locationTag attribute, and import the information back into the TADDM database.

1.2 Why use location tagging?

As already mentioned, the location tagging feature supports two different purposes:

- Credential isolation in shared TADDM environments.
- Automated population of the locationTag attribute for all discovered CIs.

By associating your IT resources with a specific location, and store this information as an attribute of the individual CI you can use the location in throughout the TADDM environment as a qualifying attribute when you create Collections, rule-based Business Applications, or reports.

The benefits you achieve by populating the locationTag attribute is, that you can use the information in the locationTag attribute for example to:

- Differentiate environments (customer, development, test, staging, production) by location association.
- Create static collections that are populated based on location association.
- Build business application definitions that are environment specific.
- Provide reports that are specific for particular environments.
- Associate locations with sites, contacts, and owners to easily obtain administrative information by location.

In addition, you may realize additional value of maintaining the content of the locationTag attribute because you can leverage information in:

Event Management	Route an event to a certain support group based on location.
Request Management	Filter available resources for which a service is requested based on the location.
Asset Management	Which physical devices are located in Thule? Are we insured?
Change Management	Which resources are affected when a generator needs to be replaced in Ballerup?
Incident Management	Who is the on-site contact for the server in Reykjavik?
Problem Management	Has anything changed lately in Torshavn?
Release Management	Update all the Windows servers in Lulea.
Site Management	Which resources are located in Server Room 11 in Oslo, which is about to be shut down because of loss of cooling.
Dash boarding	What are the main issues for the last 2 hours in Sandvik?
Service Management	Are we in compliance with the SLAs that are related to Bergen?

It goes without saying, that these capabilities are very useful when using TADDM to manage a multi-customer environment, or when your business processes require that you can distinguish resources or environments based on the physical location.

1.3 Static and Dynamic location tagging

Dynamic and static location tagging are the only types of location tagging that are supported by the built-in location tagging feature These types of location tagging are only available if the location tagging feature has been enabled on the TADDM Discovery Server. When enabled, the built-in location tagging feature automatically sets the value of the locationTag attribute for all discovered components.

Whether you use dynamic or static location tagging, you must remember that the location is used to determine which credentials are used for the discovery. These credentials are used to access both anchor hosts and target systems to be discovered. The impact of this is, that unless you ensure that the appropriate credentials are defined for the locations you have configured into the system (*static*), or the location you specify at run-time (*dynamic*), chances are that you will experience discovery errors related to missing credentials.

1.3.1 Dynamic location tagging

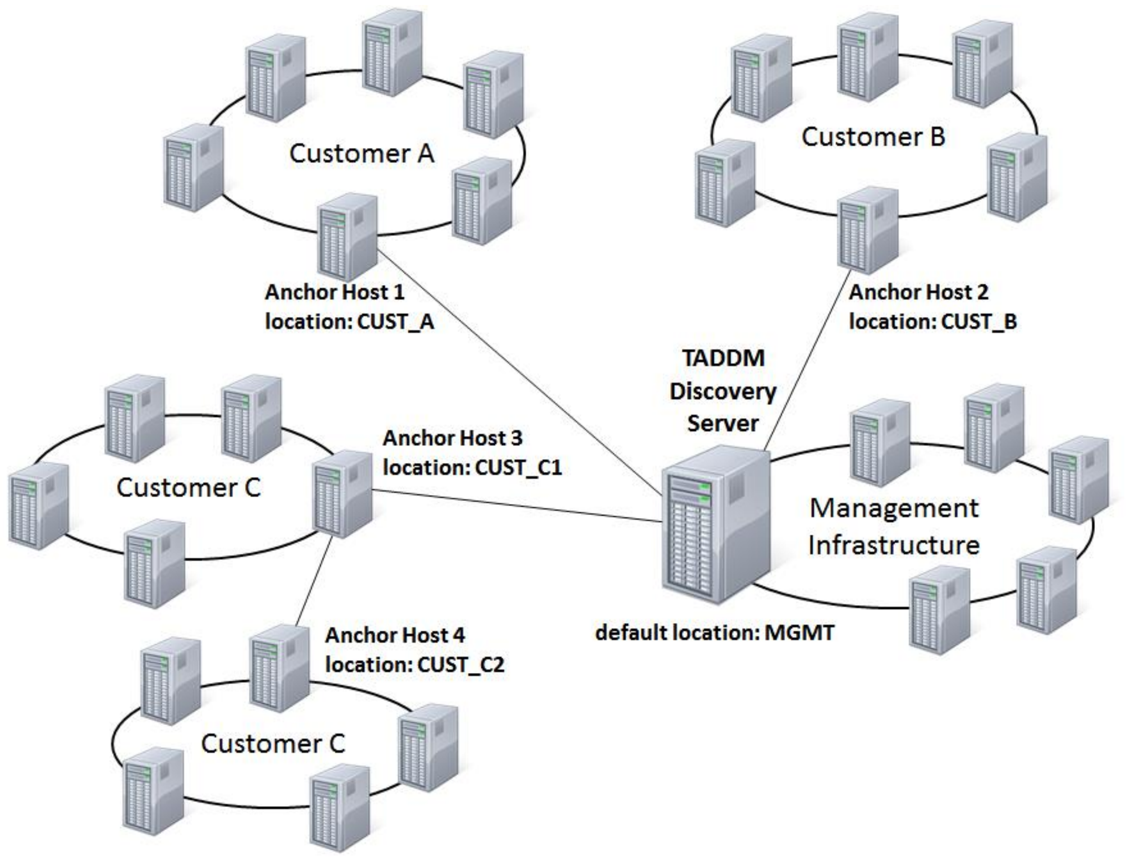
When applying the `–l <location>` argument to the `api.[bat|sh]` command used to initiate a discovery you can provide the name of the location to be used. This implies, that the discovery engine will only use credentials defined for that location, and that the value of the locationTag attribute for all discovered components is set to the specified location. The full syntax of the command to start a discovery using dynamic location tagging is:

```
$COLLATION_HOME/sdk/bin/api.sh –u <user> –p <password>
–H <Taddm_Discovery_Server> -P <Taddm_Discovery_Server_Api_Port>
discover start –l <location> [--profile <DiscoveryProfile>] <scope>
```

Dynamic location tagging is also enabled for the `loadidml.[bat|sh]` command. This allows you to assign a specific value to the locationTag attribute of components that are created during import of a discovery library book.

1.3.2 Static location tagging

Static location tagging is based on the location associated with the anchor host that performs the discovery of your components. By providing a location value for each anchor host, this value is used to populate the locationTag attribute for the components discovered through the anchor host.



In the example above, you see four anchor hosts. If you do not configure a location for any of these, the value of the locationTag attribute will be set to the default location - MGMT - (from the TADDM Discovery Server) for all discovered attributes.

If you configure the location of each of the anchor hosts, the locationTag attribute of any component discovered through an anchor host will be assigned the value of the location assigned to the anchor host. If an anchor host has no location assigned, the location tag of the discovered components will populated with the value of the default location as it is defined to the TADDM Discovery Server.

If you have multiple TADDM Discovery Servers, and do not configure specific locations for each acnhor host, the default location of each TADDM Discovery Server is applied to all the components discovered by the Discovery Server. In TADDM streaming environment using customer specific discovery servers, this may be the easiest way to populate the location information.

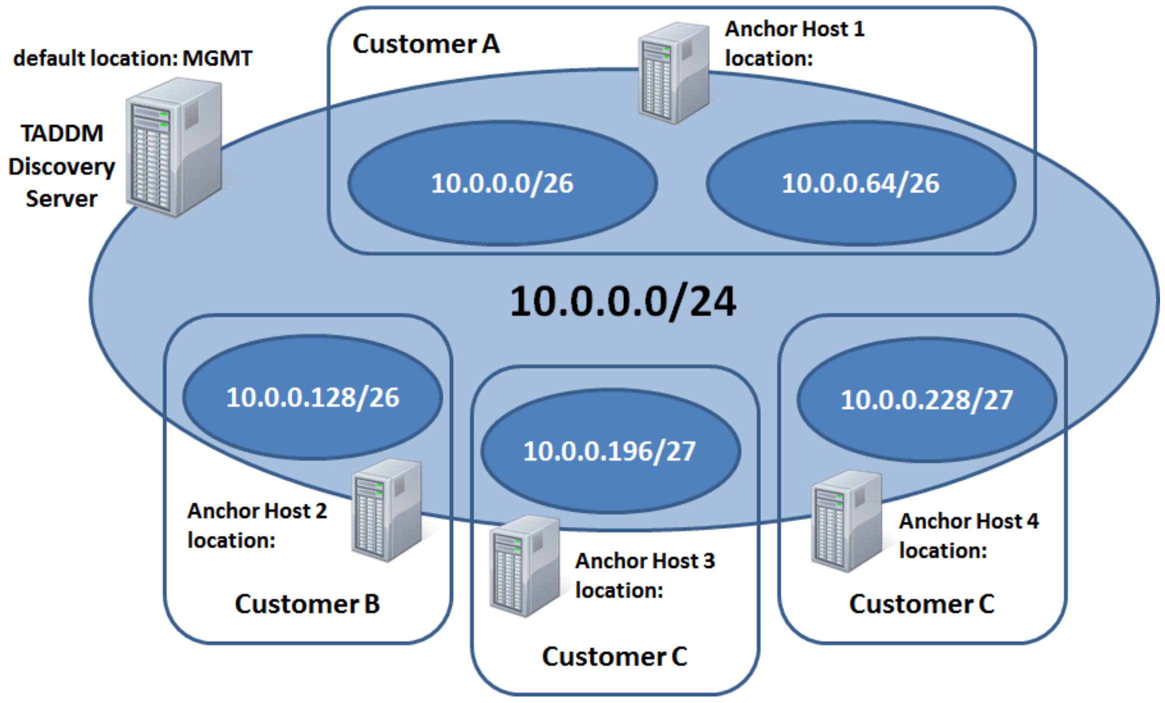
The value assigned to the locationTag attribute for all components is the location assigned to the anchor host through which the components are discovered. This implies that the locationTag for the three remote anchor hosts labled *Anchor Host 1 -3*, will be populated with the value of the default location - MGMT - because they are discovered by the local anchor on the TADDM Discovey Server. In a similar fashion, the locationTag attribute for *Anchor Host 4* will be assigned the location value of the upstream anchor host. In this case it will be *CUST_C1*.

1.4 Location tagging and scope restrictions

As you have seen, the introduction of the location tagging feature provides an extra facility to control which credentials apply to a specific system. The location is the primary qualifier used to determine which credentials to use, but for all credentials belonging to the same location, you can still apply scope restrictions to control which credentials are used for which target systems. Scope-restrictions for credentials have been a standard feature in TADDM since its conception.

Scope restrictions for anchor hosts, in order to control which parts of the infrastructure is discovered by which anchor hosts, is also standard part or TADDM. When using static location tagging, which uses the location assigned to the anchor host that discovers your infrastructure components, it becomes more important to control which anchor hosts support which parts of the infrastructure.

Imagine the following infrastructure:



To support this environment, the following scope sets definitions and anchor host assignements should be used to control which anchor hosts are used to discover specific parts of the infrastructure:

Name	Members	Anchor host	Description
Anchors	The three anchor hosts that the local anchor is communicating with directly. Anchor Host 1-3.	Local anchor	Required in order to restrict the use of the local anchor to support only remote anchor hosts.
Customer A	Subnet: 10.0.0.0/25	Anchor Host 1	Includes only IP Addresses that are related to Customer A.
Customer B	Subnet: 10.0.0.128/26	Anchor Host 2	Includes only IP Addresses that are related to Customer B.
Customer C-1	Subnet: 10.0.0.196/27 Anchor Host 4	Anchor Host 3	Includes IP Addresses that are related to Customer C. in order to pass discoveries through Anchor Host 3 to Anchor Host 4, Anchor Host 4 must be included in the scope set that is assigned to Anchor Host 3.
Customer C-2	Subnet: 10.0.0.228/27	Anchor Host 4	Includes IP Addresses that are related to Customer C's second location.

If your environment allows multiple anchor hosts to access the same target systems you cannot control which anchor is used to discover your resources. When using static location tagging, you do not know which location is used to find the credentials to be used, so unless you apply scope restrictions to your anchor hosts, you must apply the same location to the anchor hosts, or replicate the credential definitions so they exists in the context of all possible locations. For that reason it is highly recommended that you assign scope sets to your anchor hosts to control which anchor host is used to discovery specific target systems, and thereby control the location context of the discovery and the value that eventually will be populated into the locationTag attribute for the discovered components.

In addition to these scopes, you would probably create a number of supporting scope sets that are used to control the use of credentials. To control which credentials TADDM attempts to use to access target systems, you should consider using scope restrictions so only a specific subset of your credentials are used when the anchor hosts connect to the components to be discovered. The additional scope sets you need to support the sample environment are:

Name	members	Description
All	Subnet: 10.0.0.0/24	
Customer C	Subnet: 10.0.0.196/26	Includes all IP Addresses that are related to Customer C – and well as the remote anchor host Anchor Host 4.

By defining a scope set that includes all of Customer C’s target systems, you can restrict all Customer C specific credentials the the same scope, and will not have to maintain duplicate the definitions for each subnet. The All scope set includes all the IP addresses in the infrastructure, and provides the means for you to all associate global credentials with the entire subnet. Global credentials may be used to provide credentials that are related to managing the environment (for example to access anchor hosts) and therefore are not customer specific.

1.4.1 Dynamic location tagging and credential restrictions

A vital point to remember is, that when location tagging is enabled, only credentials that are related to the discovery location context will be used. If you do not configure a default location for the TADDM Discovery Server, you will see a large number of warning messages in the log files, so when enableling location tagging for the first time, you should both ensure that:

- The default location has been specified for the TADDM Discovery Server
- All your existing credentials are associated with the default location.

Assume for a while, that you use dynamic location tagging, or have not applied specific locations to the anchor hosts. In this case, all the credentails that are used in the discovery <u>must</u> either

- be related to the specific location specified on the `discovery start –l <location>` command (dynamic location tagging)

or

- belong to the defalt location as it has been configured for the TADDM Discovery Server (when no location is provided when launchng the discovery).

To be able to control which credentials are used for the discovery of the three customer environments, you can apply scope restrictions. In this case you would reuse the scope sets that were defined to control the use of anchor hosts, and restrict the credentials to the scope sets that apply to a specific customer. You can also use global credentials, that are used for specific resources in all three customer environments. Like the other credentials, the common credentials <u>must</u> be related to the location context of the discovery.

Assuming that you :

- have associated all credentials with the default location (MGMT).
- have restricted credentials in accordance with the customer scope sets.
- have not provided location configuration for any anchor hosts.

If you, under these conditions, perform a standard level 2 discovery of your environment from the TADDM Discovery Management Portal, the location context shown below will be used to identify credentials and populate the locationTag attributes:

TADDM Discovery Server configuration					
Location tagging		enabled			
Default location		MGMT			
Discovery parameters					
Location (-l)					
Scope		**All**			
Anchor host configuration					
Name	**Scope**	**Location**	**Location context used to select credentials**	**Credentials can be associated with a scope set**	**Value used to populate the locationTag attributes**
Local anchor	Anchors	n/a	MGMT	Anchors	MGMT
Anchor Host 1	Customer A		MGMT	Customer A	MGMT
Anchor Host 2	Customer B		MGMT	Customer B	MGMT
Anchor Host 3	Customer C-1		MGMT	Customer C	MGMT
Anchor Host 4	Customer C-2				

As you can see, because no anchor host location information is provided, the default location is used to identify all credentials and populate the locationTag attributes.

If you launch the discovery from a command line, and provide a location (`discover start -l <location>`) the specified location will be used instead of the default location.

1.4.2 Static location tagging and credential restrictions

To be able to differenciate the locations for customers A, B and C, you must specify the location for the anchor hosts. This will not only affect the value that is assigned to the locationTag attribute for the discovered components, but also affect the way credentials are being selected.

During discovery, TADDM uses the location specified for the anchor host to determine the location-context. This location-context is then used to select the credentials that can be used for discovering a particular component. The selection is based on the component type, and the location. Once a number of potential credentials have been identified, TADDM uses the standard scope restriction process to filter out credentials that do not support the IP Address of the component that is being discovered, and finally tries to connect to the component using each of the remaining credentials one-by-one.

Assuming that you have configured locations for the anchor hosts similar to the information below, the table shows the location context that will be used to identify credentials and populate the locationTag attributes if you perform a standard level 2 discovery of your environment from the TADDM Discovery Management Portal:

TADDM Discovery Server configuration					
Location tagging		enabled			
Default location		MGMT			
Discovery parameters					
Location (-l)					
Scope		**All**			
Anchor host configuration					
Name	**Scope**	**Location**	**Location context used to select credentials**	**Credentials can be associated with a scope set**	**Value used to populate the locationTag attributes**
Local anchor	Anchors	n/a	MGMT	Anchors	MGMT
Anchor Host 1	Customer A	CustA	CustA	Customer A	CustA
Anchor Host 2	Customer B		MGMT	Customer B	MGMT
Anchor Host 3	Customer C-1	CustC-1	CustC-1	Customer C	CustC-1
Anchor Host 4	Customer C-2	CustC-2	CustC-2	Customer C	CustC-2

Notice that if no location is specified for an anchor host, the default location is used. In the table above, you also see that the same location can be applied to multiple anchor hosts.

1.5 Summary

As you understand, the location tagging feature provides two key functions:

- Credential isolation between different parts of the TADDM infrastructure
- Automatic population of the locationTag attribute for all discovered components

Before you enable the location tagging feature you should consider if the additional administrative burden related to maintaining credentials outweighs the benefits. If security, and credential isolation is your key concern, the choice is obvious. The same is true if you are looking for a solution that automatically maintains the locationTag for all discovered components. You should also revise your credentials setup and see if you current scope restrictions can be mapped to locations.

However, if you are looking for a solution that only populates the locationTag attribute for ComputerSystems or specific Custom Application Servers, you should consider using template extensions. By applying your own extensions to the TADDM templates you will not have to maintain credentials for each unique location, and if you base your extension on a general script, the maintenance of the solution is minimized.

Notice, that if location tagging is enabled, the value of the locationTag is maintained by the location tagging feature and will overwrite any values provided by your extensions.

2 Configuring and using location tagging

The built-in location tagging provides two fundamentally different types of location tagging: Dynamic and Static. These features are only active if location tagging has been enabled on the TADDM Discovery Server.

2.1 Dynamic location tagging

When you provide a specific location tag as argument to a command line discovery or a bulkload, this specified value will be used to populate the locationTag attribute of all the discovered components. Notice that this option is restricted to command-line invocation.”

To use dynamic locationTag assignment you must ensure that location tagging has been enabled in the configuration of the TADDM Discovery Server, and add the `–l <locationTag>` option to the discovery or bulkload commands. In the following examples, the location assigned will be *From Discovery* or *From DLA* respectively:

Discovery - must be run against a TADDM Discovery Server:”

```
$COLLATION_HOEM/sdk/bin/api.sh –u administrator –p collation
–H <TADDM_discovery_server> -P 9530 discover start –l “From Discovery” –
profile “level 2 discovery” CustomerA CustomerB CustomerC
```

In order for this discovery to work, you must have assigned the *From Discovery* location to the credentials that should be used to access the components to be discovered.

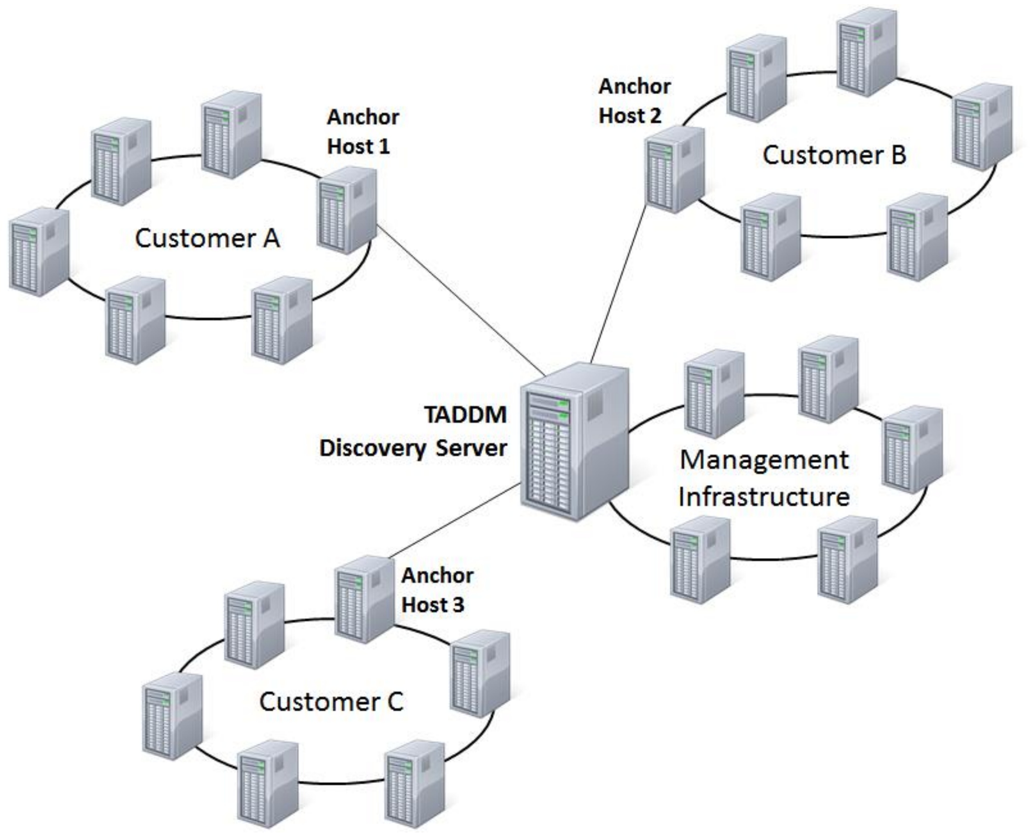
Bulk load - must be issued from a TADDM Storage Server:

```
$COLLATION_HOME/bin/loadidml.sh –u administrator –p collation
–f <location of the discovery library book> -l “From DLA”
```

Since credentials are not used when loading IdML files into the TADDM database, you can supply any value. However, all values for the locationTag attribute in the file will be overwritten by the location tagging feature.

2.2 Static location tagging

Static location tagging is based on the anchor host that performs the discovery of your components. By providing a location value for each anchor this value is used to both identify the credentials to be used for the discovery, and populate the locationTag attribute for the components discovered through a specific anchor host.



In the example above, you see three anchor hosts. If you configure the location of each of these anchor hosts in the `$COLLATION_HOME/etc/anchor.properties` file on the TADDM Discovery Server, and enable location tagging, the components located in the three remote subnets will be asigned the location of their local anchor host the next time they are discovered. The components discovered by the local anchor host on the TADDM Discovery Server will be assigned the default location that is defined in the *com.ibm.cdb.locationTag* property in the `$COLLATION_HOME/etc/collation.properties` file on the TADDM Discovery Server.

Note: Because the location tagging is directly related to the anchor hosts, when using dynamic location tagging, all components disovered asynchronously or through the ITM infrastructure will be assigned the default location as it is defined in the *com.ibm.cdb.locationTag* property.

2.3 Configuring dynamic and static location tagging

To enable location tagging you must modify the configuration of your TADDM Discovery Servers in the `collation.properties` file. If you wish to use the static location tagging facility, you must also provide location information for the anchor hosts defined in the `anchor.properties` file on the TADDM Discovery Servers.

2.3.1 Enabling automatic location tagging

To enable the automated location tagging, all you have to do is to set the value of the *com.ibm.cdb.location taggingEnabled* property to `true` in the `collation.properties` file that controls the configuration of the TADDM Discovery Servers in your environment. When the server is restarted, the location tagging is active.

Note that if you enable location tagging, and do not provide a value for the *com.ibm.cdb.locationTag* property, you will see a large number of warning messages indicating that the default location has not been specified. These massages appear in the log files when you perform discovery. The format of the messages is similar to this example:

```
util.LocationTagResolver - Warning: invalid ModelFactory.newInstance usage
��� locationTag not set in properties but location tagging is enabled
```

2.3.2 Configuring locations

You define anchor locations and the default location in two separate locations on the TADDM Discovery Server. The *default location*, which is assigned to components discovered through anchor hosts that have no specific location assigned, is defined in the TADDM Discovery Server configuration, and locations for individual anchor hosts are maintained in a separate file.

2.3.2.1 Default location

The default location is defined in the `$COLLATION_HOME/etc/collation.properties` file on each TADDM Discovery Server and is specified in the *com.ibm.cdb.locationTag* property. This property is not pre-defined in the file, and has no system provided default value.

To set the value of default location to *MGMT*, add the following line to the `collation.properties` file.

```
com.ibm.cdb.locationTag = MGMT
```

To activate your updates, the TADDM Discovery Server must be restarted.

2.3.2.2 Anchor host locations

To specify the location for each anchor hosts in your environment, you must update the `$COLLATION_HOME/etc/anchor.properties` file. This implies that you must first define the anchor hosts to the environment using the Discovery Management Console, and then you can update the definitions in the file.

To add the location to an anchor host definition in the `anchor.properties` file, add an *anchor_location_N=<location>* line to the definition. For example, to set the location to *CustC-1*, for the third anchor host in your environment, add the following line:

```
anchor_location_3 = CustC-1
```

If the `anchor_location_N` specification is not added for one or more anchor hosts, the default location will be used. The following example shows how to set the locations for the three anchor hosts depicted in the sample topology:

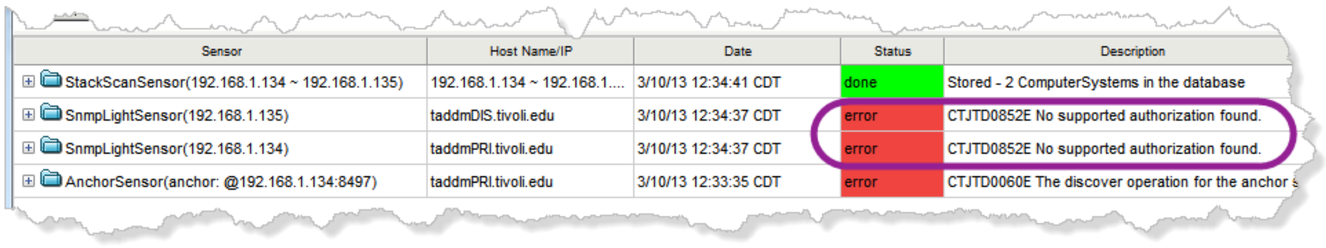
```
anchor_host_0=root server
anchor_scope=Mgmt
#
anchor_host_1=10.0.0.100
anchor_scope_1=Customer A
anchor_location_1=CustA
#
anchor_host_2=10.0.0.130
anchor_scope_2=Customer B
anchor_location_2=CustB
#
anchor_host_3=10.0.0.200
anchor_scope_3=Customer C-1
anchor_location_3=CustC
Port=8497
#
anchor_host_4=10.0.0.200
anchor_scope_3=Custtomer C-2
anchor_location_3=CustC
Port=8497
```

Notice that no location is assigned to the local anchor host - `anchor_host_0` - which resides on the TADDM Discovery Server. As shown for anchor_host_3 and anchor_host_4 you can see that the same location can be applied to different anchor hosts.

If you apply the following anchor configurations, you must re-configure your access control lists to support the locations you have assigned to the anchor hosts. If you fail to do so, your discovery will not be able to identify the credentials to be used to access the components to be discovered, and thus, the discovery will fail.

3 Configuring credentials

When you perform your first discovery that requires credentials (snmp-based level 1, level 2 or level 3 discovery) after having enabled the location tagging feature, you will most likely experience that your discoveries will fail. You will see messages similar to this:



Sensor	Host Name/IP	Date	Status	Description
StackScanSensor(192.168.1.134 ~ 192.168.1.135)	192.168.1.134 ~ 192.168.1....	3/10/13 12:34:41 CDT	done	Stored - 2 ComputerSystems in the database
SnmpLightSensor(192.168.1.135)	taddmDIS.tivoli.edu	3/10/13 12:34:37 CDT	error	CTJTD0852E No supported authorization found.
SnmpLightSensor(192.168.1.134)	taddmPRI.tivoli.edu	3/10/13 12:34:37 CDT	error	CTJTD0852E No supported authorization found.
AnchorSensor(anchor: @192.168.1.134:8497)	taddmPRI.tivoli.edu	3/10/13 12:33:35 CDT	error	CTJTD0060E The discover operation for the anchor

The message *CTJTD0852E No supported authorization found* indicates that TADDM did not find any credentials in the access list that are applicable to the specific system that is about the be discovered. This may seem strange since you previously have been able to discover, and the credentials have not changed.

The reason for this behavior is that when location tagging is enabled, only credentials that are associated with the location which has been assigned to the anchor host that supports the selected scope (or the dynamically supplied location), can be applied. To find credentials to use to access the target components the TADDM discovery performs the following steps:

1. The IP address of the target component is identified either directly from the target of the discovery or by performing a DNS lookup.

2. The scopes that include the target component IP address are identified.

3. The location to be associated with the target system is determined. This determination can be performed in three different ways:

    - If you are using dynamic location tagging, by specifying the `-l` argument to the command-line discovery the location value provided on command invocation is assigned to all target components.

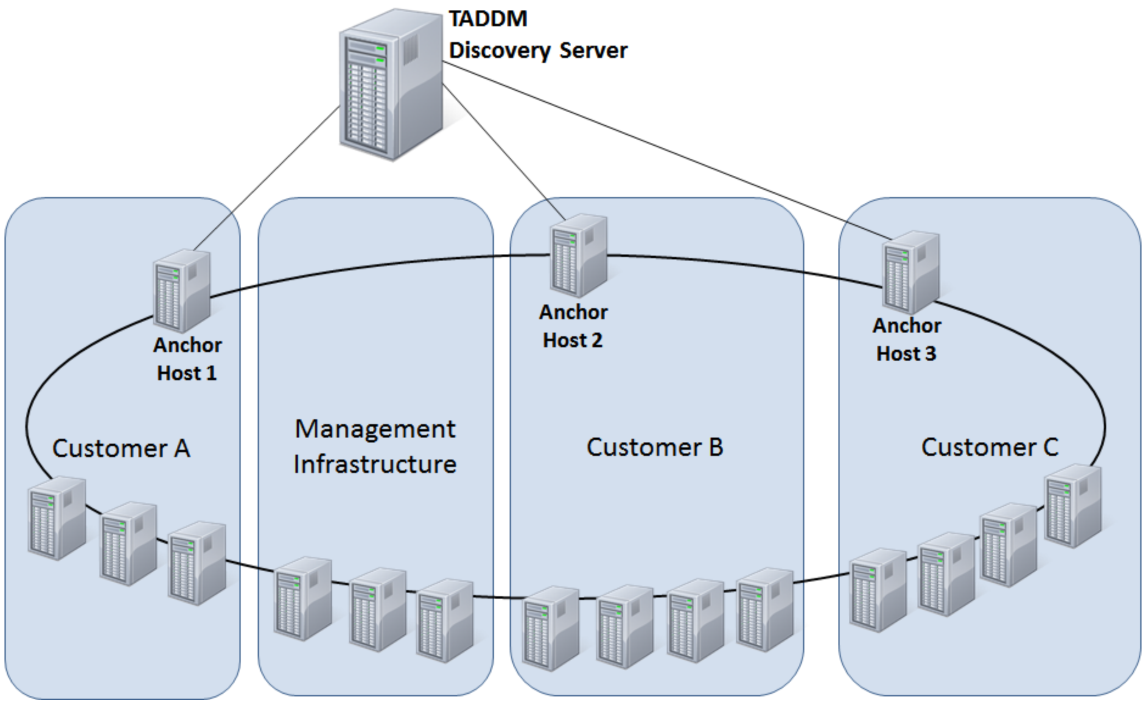
    - If the target component IP address is included in a scope that is associated with a specific anchor host, TADDM assigns the location associated with the anchor host (from the `$COLLATION_HOME/etc/anchor.properties` file) to the target component.

    - If an anchor location cannot be determined, TADDM assigns the default location to the target component. The default location is defined in the *com.ibm.cdb.locationTag* property in the `$COLLATION_HOME/etc/collation.properties` file.

4. When the location has been identified, TADDM uses the type of the target component, the IP address, and the location to find candidate credentials in the access list. To qualify as a candidate credential, credentials must:
   - Be associated with the location that is identified for the target component.
   - Have a type that applies to the target component that is about to be discovered.
   - Support the IP address of the target component in the associated scope restriction (if any)
5. When candidate credentials are identified, TADDM first inspect the credentials specified directly in the active discovery profile, and if access to the target component is not established, TADDM inspects the global credentials to find credentials that can be used to successfully access the target resource.

As you understand, location tagging adds an extra, mandatory qualifier that is used to identify candidate credentials. You may argue that this makes credential management more cumbersome, but that is the price you pay for enhanced security provided by the credential isolation function of the location tagging feature.

The benefits of this behavior are primarily realized in TADDM infrastructures in which you need to isolate the credentials between different business units or customers. By grouping credentials based on location, you prevent that credentials that apply to a unique customer or business unit are revealed to extension scripts or sensors that operate in the context of another location. This means that you can host components from multiple customers in the same scope (subnet) without risking exposing credentials across customer boundaries.



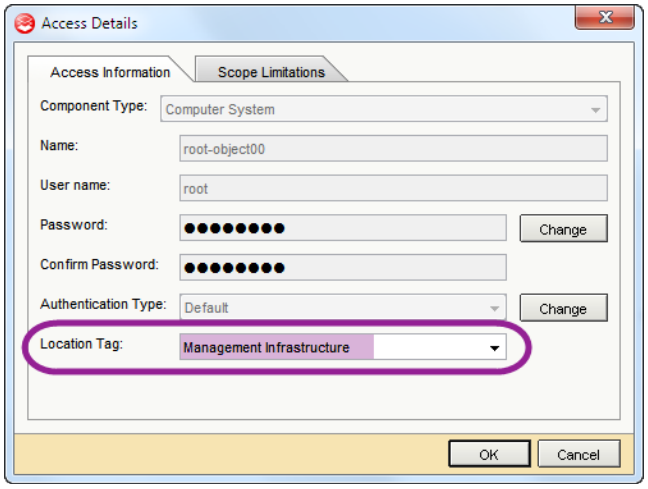
However, the drawback is that to harvest the benefits using static location tagging, you must deploy customer specific anchor hosts. If you do not want to do this, you must start discoveries using the command-line API and supply the specific location for your discovery. The location is defined as an attribute to the anchor host that performs the discovery, so to select the correct credentials the anchor hosts must support only scopes for a specific location (or customer). Using the command-line to launch your discoveries you can specify the location when starting a discovery, and thereby force the use of location-specific credentials. Unfortunately this feature is not available when defining schedules or initiating a discovery from the Discovery Management Console, so for those use cases, you must use static location tagging.

3.1.1 Sharing TADDM infrastructure components

If you want to share TADDM infrastructure components such as anchor hosts and windows gateways across multiple locations, you must use dynamic location tagging. This implies that you must launch all your discoveries from a command line and remember to apply the appropriate location as an argument to the `discover start` command. It goes without saying, that credentials for the supplied location must have been defined in advance. For all practical purposes this implies that you must replicate the credentials for your shared components so that TADDM can find valid credentials for all the locations for which you wish to share the infrastructure components. So, by enabling location tagging you accept that extra effort must be put into maintenance of credentials that apply to TADDM infrastructure components that are shared across locations.

3.2 Assigning locations to Access Control Lists

To assign a location to your access control lists, you must edit the access control lists from the Discovery Management Console. When location tagging is enabled, you will see a new field named *Location Tag* in the Access Details dialog.



Component Type:	Computer System
Name:	root-object00
User name:	root
Password:	●●●●●●●●
Confirm Password:	●●●●●●●●
Authentication Type:	Default
Location Tag:	Management Infrastructure

To assign a location to the credential, simply apply a value in the Location Tag field, or select one from the drop-down list. As you might have expected, the value of the Location Tag is case sensitive. To make sure that you provide the same value as was assigned to the anchor hosts you can either look in the `$COLLATION_HOME/etc/anchor.properties` file, or check the output from the `$COLLATION_HOME/bin/healthcheck.jy checkTaddmAcnhors` utility. No matter which method you choose, you must have access to the TADDM Discovery Server.

```
taddmusr@taddmDIS:/opt/IBM/taddm/dist/bin> ./healthcheck.jy checkTaddmAnchors

GROUP:  config
*************************************************************************
**                  Begin checkTaddm Anchors                           **
**                  ------------------------                           **
** This section displays information about the Taddm Anchor Servers    **
*************************************************************************
Label                      Value
anchor_location_3     CustC
anchor_location_2     CustB
anchor_location_1     CustA
port                        8497
anchor_host_3         9.33.12.12
anchor_host_2      192.168.1.134
anchor_scope_3             Customer C
anchor_host_1       9.48.132.254
anchor_scope_2             Customer B
anchor_scope_1             Customer A
```

When you have associated all your access control lists with a location, you are ready to start discovering and associating all your discovered components with the location.

4 Populating the locationTag attribute without using the location tagging feature

As mentioned you can easily develop your own template extensions to populate the locationTag attribute for component types that are associated with templates. However, if you do, you must **disable the location tagging feature** since it takes precedence over your extensions. In TADDM version 7.2.1 only ComputerSystems and Custom Application Servers support template extension commands.

Compared to using the built-in location tagging feature, using template extension commands does require a bit more administrative work. The TADDM administrator must apply extension commands for all the types of resources for which you want to populate the locationTag attribute. Even if you only want to assign values to the locationTag attribute for ComputerSystems you must define and maintain multiple extension command definitions. However, you can ease maintenance by referencing common extension scripts from your extension commands. This way, all the template extension commands will be identical and they all call the same extension script which support all the different types of operating systems in your environment.

In the following three ways to create your own extensions to populate the locationTag attribute are demonstrated:

1. Using extension commands to apply dynamic values to the locationTag attribute.
2. Using extension commands to apply static values based on your own rules.
3. Using extension scripts to emulate the behavior of the static location tagging feature.

Note: Using custom extensions to populate the locationTag attribute is only supported for component types that are supported by a template. This means, that you can only use this technique to populate the locationTag for ComputerSystem and AppServer components. You *can* create you own bach script to propagete the locationTag of, for example a ComputerSystem to all the components it hosts, but instead you should consider using the built-in location tagging feature.

In contrast, the built-in location tagging feature populateds the locationTag attribute for all discovered components.

4.1 Populating the locationTag attributes from template extensions

To apply a dynamic location to the discovered component types that support template extensions, you extend the templates in the usual manner by creating a file in the `$COLLATION_HOME/etc/templates/commands` directory on the TADDM Discovery Server and name the file after the template you wish to extend. If, for example, you wish to extend the *LinuxUnitaryComputerSystem* template, the fully qualified path of the file you create on the TADDM Discovery Server must be:

```
$COLLATION_HOME/etc/templates/commands/LinuxUnitaryComputerSystem
```

In this file you supply control statements to instruct TADDM how to populate attributes based on commands that will be executed on the target system during discovery. To assign a dynamic value of *MGMT* to the locationTag attribute you would provide the following line:

```
CMD:locationTag=echo MGMT
```

This will set the value of the locationTag of all LinuxUnitaryComputerSystems discovered from your TADDM Discovery Server to *MGMT*.

Naturally, the command that executed on the target system can be a bit more advanced, but you are limited to a single line. The following example assigns all but the last two qualifiers of the IP domain name of the Linux systems to the locationTag attribute:”

```
CMD:locationTag=`hostname –d | awk ‘BEGIN{FS=”.”} {for(i=1;i<NF-2;i++)
if(i==1) {d=$i} else {d=d”.”$i}} END{print d}’`
```

As an alternative you can use temporary files to store the output from intermediate commands analyze the results in subsequent commands. If you feel uncomfortable creating long one-line scripts, consider using the NOP operation. The following extension commands can be used to apply specific values to the locationTag attribute based on the three first octets of the IP address:

```
CMD:NOP=echo UNKNOWN > /tmp/resultFile
CMD:NOP=echo subNet=`hostname –i | cut –d ”.” –f 1,2,3 > /tmp/workFile`

CMD:NOP=source /tmp/workFile ; if [ “$subNet” = “10.0.0” ] ; then echo CustA
> /tmp/resultFile
CMD:NOP=source /tmp/workFile ; if [ “$subNet” = “10.0.1” ] ; then echo CustB
> /tmp/resultFile
CMD:NOP=source /tmp/workFile ; if [ “$subNet” = “10.0.2” ] ; then echo CustC
> /tmp/resultFile

CMD:locationTag=cat /tmp/resultFile

CMD:NOP=if [ -e /tmp/resultFile ] ; rm –f /tmp/resultFile; fi
CMD:NOP=if [ -e /tmp/workFile ] ; rm –f /tmp/workFile; fi
```

In the example above, the CMD:NOP operation is used to manipulate files that contain the information that eventually is populated into the locationTag attribute. The CMD:NOP operator is also used to clean up after a value has been assigned to the attribute. The example above could have been implemented in many other ways, for example by using the CMD:NOP operator a number of times to write a shell script to a temporary file, and then execute the script as part of the *CMD:locationTag* operator.

When a discovery is performed using a profile in which the GenericComputerSystemSensor or the CustomAppServerSensor is enabled, the template extension will be invoked for all the components that matches the template. This means that you cannot use this technique to populate the locationTag attribute of sensor-supported resources, such as WebSphere or DB2 servers. However, this restriction may be lifted in a future release of TADDM.

To overcome the restriction, you can relatively easily create a script that uses the TADDM APIs to propagate the location of the ComputerSystem to all the components that are hosted on the ComputerSystem. To easily find the children of the ComputerSystem, consider running the `explicitrel` script to generate explicit relationships for all the implicit relationships in your environment. This way, your solution to propagate the location can use the Relationship objects to identify components to be updated.

4.2 Using extension scripts to populate the locationTag attribute

Instead of providing a complex set of command that contain the necessary parsing to extract the exact information that needs to be populated into the attribute in the CMD:<attribute> tag, you can use a script.

Template extensions can also be implemented as JavaScript or jython scripts. Using these scripts, you must include the sensorhelper class which provides several facilities that makes your life easier. The main benefits of using external scripts are:

- You can execute multiple commands on the target system, and control the parsing in your script. This is helpful if the output from one command is used to determine what actions to take next.
- You can create new resources, and relate them to the create target resource. For example, if you discover an application server, you can discover software modules, and create the relevant SoftwareModule resources in the TADDM database.
- You can access other resources, for example databases, to obtain information. The credentials needed to access the other resources may be read from the access control list.

For more details, refer to the TADDM SDK Developers Guide.

An important thing to remember is that your extension script runs on the TADDM Discovery Server. This implies that you can also access the `collation.properties` file to obtain control information, or other files that can be accessed by the taddm user. You must also understand that from extension scripts you do not have access to the TADDM database, so you cannot query the TADDM database to reference existing resources.

In *Appendix A Location tagging.py* on page 63 you find a jython script that mimics the functionality of the built-in location tagging feature to populate the locationTag attribute of discovered ComputerSystems. In short, the functionality of the script is:

1. The script checks if the built-in location tagging feature is enabled. If it is, the script terminates, because the built-in feature always takes preference regarding the population of the locationTag attribute.
2. The script obtains the IP Address of the ComputerSystem that is being discovered.
3. The script reads the `scopes.properties` file (on the TADDM Discovery Server) to determine which scope(s) the ComputerSystem belongs to.
4. Next, the `anchor.properties` file (also on the TADDM Discovery server) is inspected to find the anchor(s) that support the scopes(s) which include the ComputerSystem IP Address
5. The locationTag attribute of the ComputerSystem is populated with the value of the anchor_location (similar to the built-in feature) from the anchor with the highest suffix that was identified in the previous step.

If the built-in location tagging feature is disabled and a value is provided for the *com.ibm.cdm.locationTag* property in the `collation.properties` file you will see a lot of warning messages in the log files. Because of this, the script uses a new property - *com.ibm.cdm.customLocationTag* - to store the default location which is assigned to ComputerSystems that are not associated with an anchor.

To configure your system to use this script to assigning locations to your Linux ComputerSystems, perform the following steps:

1. Copy the Location tagging.py script to the following directory on your TADDM Discovery Server(s):

   ```
   $COLLATION_HOME/etc/templates/commands/extension-scripts
   ```

2. Modify the `LinuxComputerSystemTemplate` file in the `$COLLATION_HOME/etc/templates/commands` directory so it only contains the following line:

   ```
   SCRIPT:etc/templates/commands/extension-scripts/Location tagging.py
   ```

   Notice that the path to the script is relative to $COLLATION_HOME.

If you want to apply the same behavior to other computer system types, simply copy the LinuxComputerSystemTemplate file to a new file that has the same name as the computer system template that identifies your computer systems. For example *AixComputerSystemTemplate* or *WindowsComputerSystemTemplate*.

The next time you perform a discovery in which both the GenericComputerSystemSensor template and the relevant computer system sensors are activated, the locationTag of the discovered computer systems will be populated.

5 Consuming the locationTag information

Most likely, you have not gone through the effort to enable location tagging in order not to use it. The three main use cases that justify enablement of location tagging are:

- Data isolation in a shared environment
- Controlling data level access based on location
- Location aware reporting
- Visualizing the location in the TADDM GUI

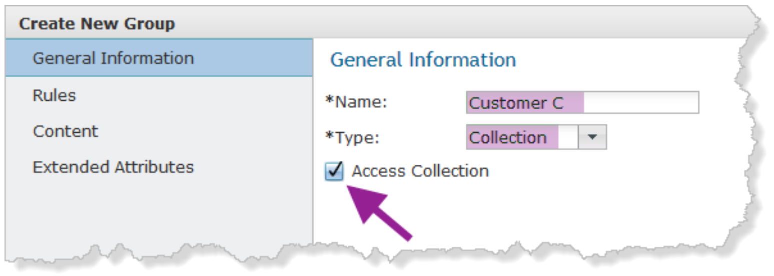
By default, TADDM does not use the locationTag attribute in any of its built-in components, so it will take a bit of work to realize the benefits it provides.

5.1 Controlling access to resources through location

One obvious use of the locationTag, especially in a multi-customer environment, is to use the locationTag to control access to components. TADDM uses access collections to control which components a user can access in the resource navigator tool, and for which resources a user can view the details.. Access collections can be defined dynamically by assigning specific discovered components as members of the collection. However, they can also be defined statically, based on an MQL query, in which case newly discovered components automatically are associated with a collection based in criteria you specify. One of these criteria can be a specific value of the locationTag attribute.

To create an access collection that can be used to allow access to the components related to the location named Customer C, you would complete these steps:

1. Open the Grouping Composer from the Data Management Portal, and create a new group.
2. Specify a name for the group, for example `Customer C`, and select `Collection` as the type for the group, and check the *Access Collection* checkbox.

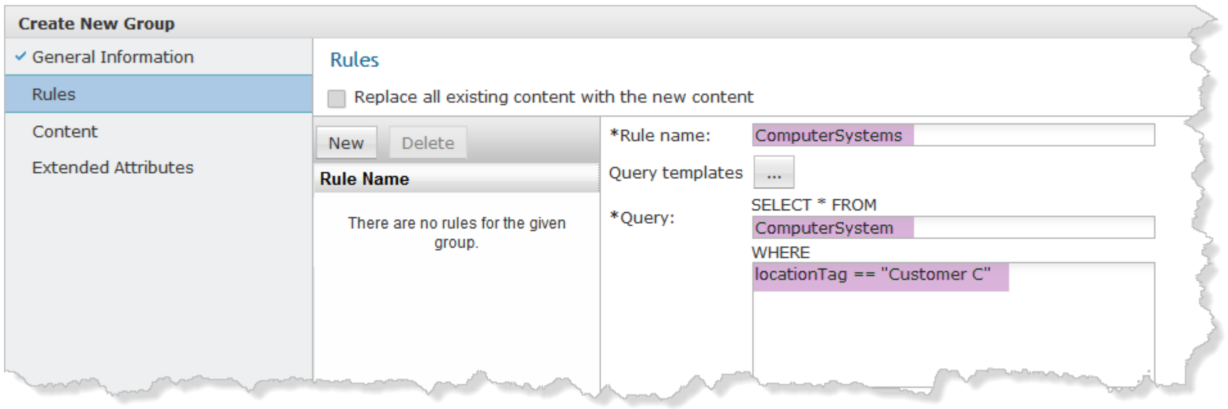


*Name: Customer C
*Type: Collection
[✓] Access Collection

3. In the Rules section of the Create New Group dialog, you apply rules for each type of component you want to include in the collection. This can be a tedious effort because of the multitude of component types supported by TADDM. However, if you apply your knowledge about the Common Data Model, and the derivation hierarchy, you realize that to include all the component types that are revealed in the Data Management Portal, all you need is rules that include the upper-most, top-level, non-transient component types. The means, that if you apply rules for ComputerSystem, ComputerSystemCluster, AppServer, AppServerCluster, Function, IPNetwork, and StorageExtent you will have covered most of the component types that are revealed in the Data Management Portal.

   To create a rule for the ComputerSystem components, supply these values:

RuleName:	ComputerSystems
Query	ComputerSystem
WHERE	locationTag ==”Customer C”

   Replace all existing content with the new content

   There are no rules for the given group.

   *Rule name: ComputerSystems

   *Query: SELECT * FROM ComputerSystem WHERE locationTag == "Customer C"

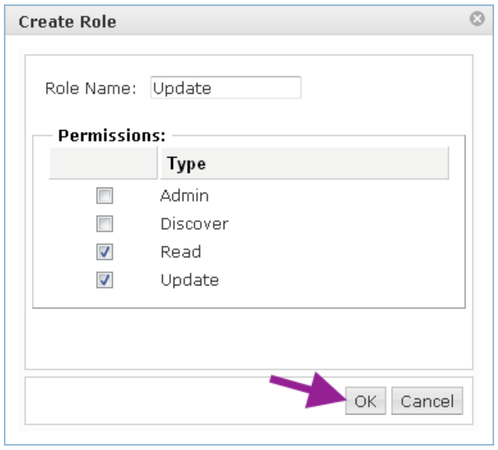
   Notice how the WHERE clause is used to include all the components for which the locationTag attribute has a specific value. All component types have an attribute named locationTag, and therefore this WHERE clause can be used for all the rules that must be applied to include all the desired component types in the collection.

Once you have created the access collection, all you need to do to enforce data isolation is to assign roles for the new access collection to the users and groups that will be allowed to access the resources that are members of the collection.

For example, to authorize the members of the *C-Customers* user group to view and update the members of the *Customer C* access location complete these steps:

1. Ensure that you have enabled data level security by setting the value of the *com.collation.security.enabledatalevelsecurity* property to *true* in the `collation.properties` file on the TADDM Primary Storage Server. If you have to change the value of the property, you must restart the TADDM Primary Storage Server before the new value becomes active.

2. Use the **Administration** > **Roles** tool to create a new role named Update. Assign only *Read* and *Update* authorization.



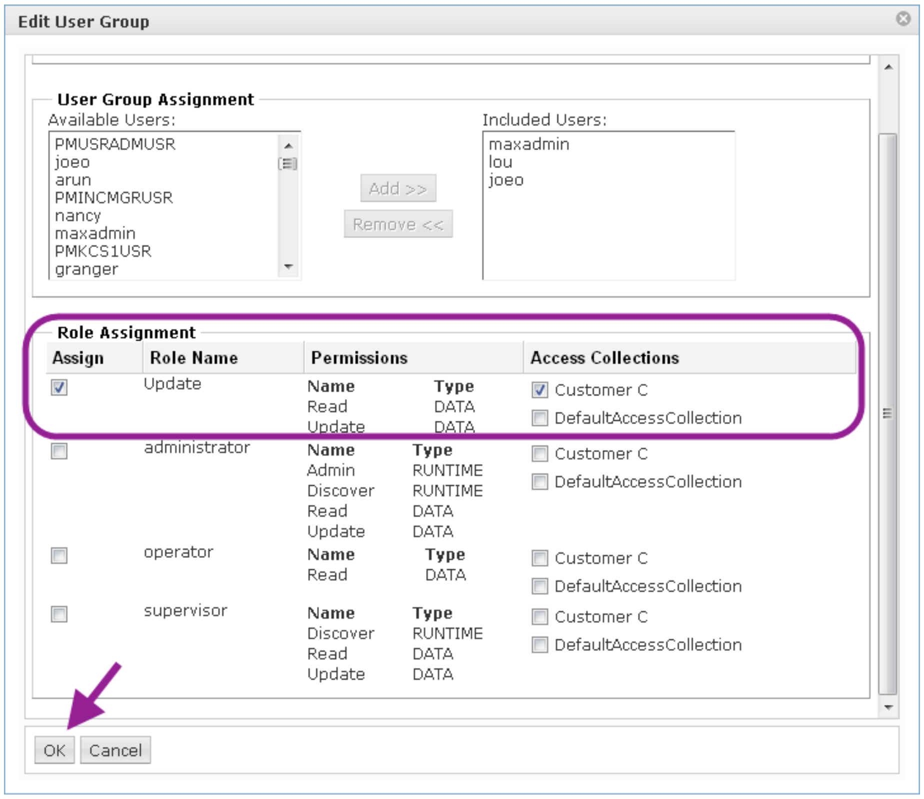
Role Name: Update

Permissions:

	Type
☐	Admin
☐	Discover
✓	Read
✓	Update

Press **OK** when you are ready.

3. To assign the new role to the group, use the **Administration** > **User Groups** tool, open the C-Customers group for edit, and assign ONLY the Customer C access collection to the update role.



User Group Assignment

Available Users: PMUSRADMUSR, joeo, arun, PMINCMGRUSR, nancy, maxadmin, PMKCS1USR, granger

Included Users: maxadmin, lou, joeo

Role Assignment

Assign	Role Name	Permissions	Access Collections
✓	Update	Name / Type: Read DATA, Update DATA	✓ Customer C; ☐ DefaultAccessCollection
☐	administrator	Name / Type: Admin RUNTIME, Discover RUNTIME, Read DATA, Update DATA	☐ Customer C; ☐ DefaultAccessCollection
☐	operator	Name / Type: Read DATA	☐ Customer C; ☐ DefaultAccessCollection
☐	supervisor	Name / Type: Discover RUNTIME, Read DATA, Update DATA	☐ Customer C; ☐ DefaultAccessCollection

When you press **OK** the read and update authorizations are assigned to all three uses in the group – but only for the resources that are members of the Customer C access collection.

When one of the users in the *C-Customers* group logs in, the only resources that the user can see are the ones that are members of the *Customer C* access collection. In this case, it will only be the ComputerSystems that are assigned the *Customer C* location. However, you can easily augment the access collection by applying a new rule that includes AppServes for which the locationTag contains the value *Customer C* – or for which the locationTag of the ComputerSystem that the AppServers run on contains the value *Customer C*.

It goes without saying that you will minimize the administration effort if you enable the built-in location tagging feature. This feature populates the locationTag for ALL discovered resources. If you use custom template extension scripts, you must manage multiple extensions, and discovered child resources will not be tagged.

5.2 Location-aware reporting

Unfortunately, none of the reports delivered with TADDM has been prepared for using the locationTag information. For details on how to use the locationTag attribute in the three distinct reporting facilities provided in TADDM, refer to the following sections.

5.2.1 Ad-hoc reports with the Custom Query wizard

The Custom Query wizard is used to build ad-hoc reports. The wizard filters in only specific attributes for each resource type, and unfortunately it does not currently (TADDM 7.2.1 FP4) include the locationTag attribute in the list. This implies that, at present, the locationTag attribute cannot be used in ad-hoc reports.

5.2.2 BIRT reports

To use the locationTag attribute in your BIRT reports, you must rely on the building-block views in the TADDM database. These are all named in accordance with this template:

```
BB_<resourceType_and_varsion>_V).
```

The building-block views all include the LOCATIONTAG_C column, which can be used to filter in (or out) resources that belong to specific locations. Remember, that the default value of the LOCATIONTAG_C field is a null value.

As mentioned, none of the system provided BIRT reports are location-aware. However, it is pretty straight-forward to add a report parameter so the user can provide the specific location to be used, and a dataset filter that used the parameter so only resources from the selected location are included in the report. The report parameter should be populated from a new dataset which selects all unique locations in your environment.

However, you may also argue, that with this new feature, you would like to build reports of all resources that have no locationTag defined, or combinations of resources belonging to specific locations and resources with no location. This can be achieved by building the where clause for the queries statically, and requires a little bit of scripting.

In the following, both the simple and the more comprehensive solution to filter your report contents based on locationTag are provided.

5.2.2.1 Simple location specific reporting

The following procedure outlines the steps necessary to make a standard BIRT report location-aware. In this example below, the TADDM Inventory report is used to illustrate the additional customization steps needed to limit the information in the report to records that are related to specific locations.

The activities needed to make the TADDM Inventory report location-aware are:

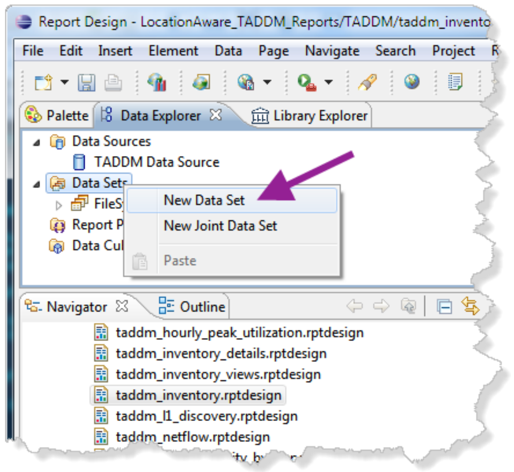
- Add a dataset that selects the unique locations from the BB_COMPUTERSYSTEM40_V building-block view.
- Create a report parameter and link it to the newly created dataset.
- Add the LOCATIONTAG_C column to the query that populates the FileSystemSet dataset.
- Add a filter to the FileSystemSet dataset, using a *LOCATIONTAG_C IN* clause, and link it to the report parameter
- Optional, add LocationTAG_C and the current value of the report parameter to the report.

Modifying the taddm_inventory.rptdesign report

To modify the TADDM Inventory report, complete these steps:

1. Start the BIRT Report Designer v2.2.1 tool, and create a new BIRT report project.
2. Imported the standard TADDM reports into the new project. The standard reports are located in the $COLLATION_HOME/etc/reporting/TADDMReports.zip on any TADDM Server.
3. Open the TADDM report named taddm_inventory.rptdesign.
4. Modify the TADDM Data Source properties so you can successfully access you own TADDM database.
5. Use these steps to create a new dataset named Locations, which is used to select the unique existing locations so they can be presented to the user:

1. From the Data Explorer pane, right-click on the Data Sets group, and select **New Data Set** from the context menu.


Report Design - LocationAware_TADDM_Reports/TADDM/taddm_invento
File Edit Insert Element Data Page Navigate Search Project
Palette
Data Explorer
Library Explorer
Data Sources
TADDM Data Source
Data Sets
New Data Set
New Joint Data Set
Paste
Navigator
Outline
taddm_hourly_peak_utilization.rptdesign
taddm_inventory_details.rptdesign
taddm_inventory_views.rptdesign
taddm_inventory.rptdesign
taddm_l1_discovery.rptdesign
taddm_netflow.rptdesign

2. Provide a name of Locations for the new data set, and click **Next**.

3. In the Query dialog, provide the following query:

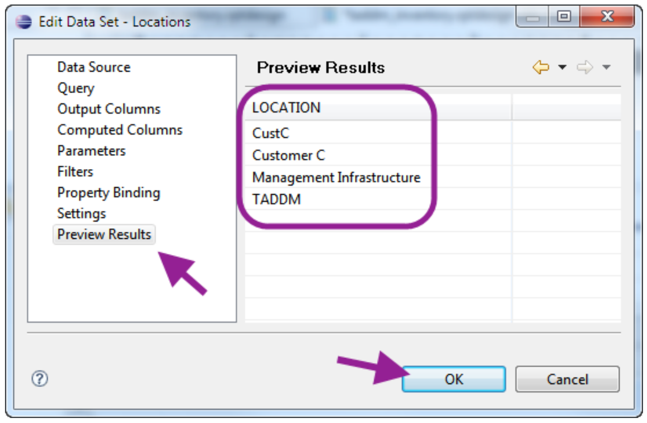
```
--select distinct(locationTag_C) as Location

--from BB_COMPUTERSYSTEM40_V

--where LOCATIONTAG_C is not NULL
```

   Click **Finish** when you are ready.

4. When the Output Columns dialog appears, click **Preview Results** and verify that you see the expected locations.



LOCATION
CustC
Customer C
Management Infrastructure
TADDM

   When you are satisfied with the result, close the dialog by clicking **OK**.

You have now created the data set that will be used to present location information to the report user.

Creating a report parameter

In BIRT report parameter can be used to prompt the user to select specific values or criteria that will be used to select content or format the report. In this example, a simple report parameter is used to force the user to select a specific location for which resources are included in the report.

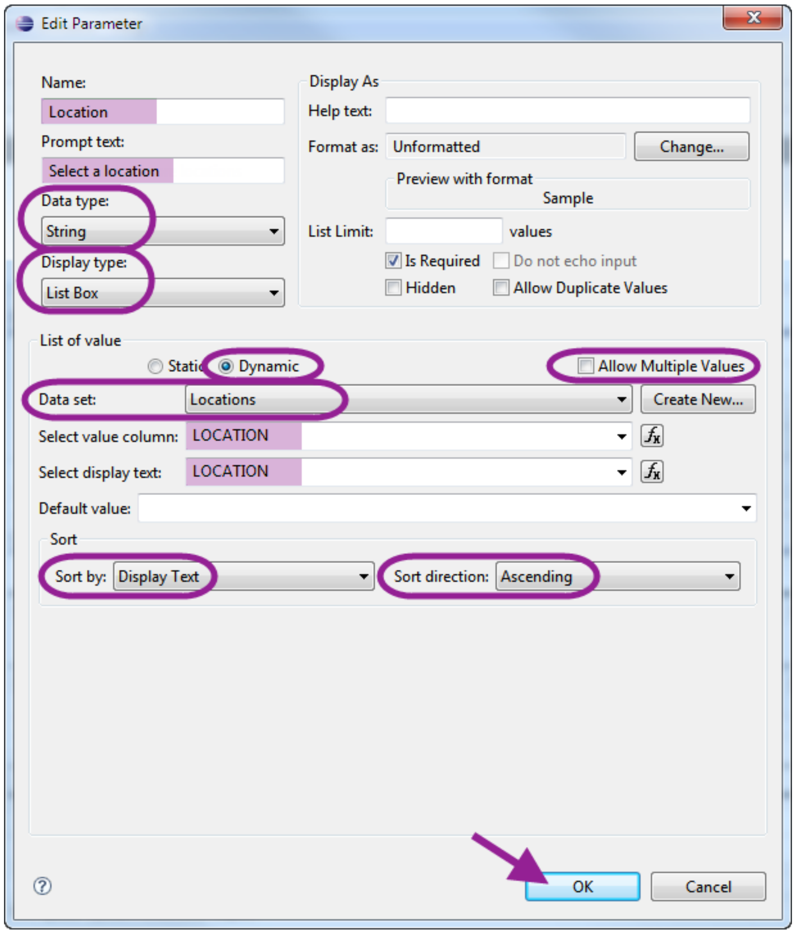
To create a new report parameter, follow these steps:

1. From the Data Explorer, right-click on the Report Parameters group, and select **New Parameter**.

2. Provide the following details for the parameter:

Name	`Location`
Prompt text:	`Select a location`
Display type	`List Box`
List of value	`Static`
Allow Multiple Values	`cleared`
Data set	`Locations`
Select value column:	`LOCATION`
Select display text:	`LOCATION`
Sort by:	`Display Text`
Sort direction:	`Ascending`

When you are done, that New Parameter dialog should look like this:



Name: Location
Prompt text: Select a location
Data type: String
Display type: List Box
Display As
Help text:
Format as: Unformatted
Preview with format: Sample
List Limit: values
[✓] Is Required
[ ] Do not echo input
[ ] Hidden
[ ] Allow Duplicate Values
List of value
Static [✓ Dynamic]
[ ] Allow Multiple Values
Data set: Locations
Select value column: LOCATION
Select display text: LOCATION
Default value:
Sort
Sort by: Display Text
Sort direction: Ascending

Notice how the parameter is linked to the dataset. This ensures that the values that the data set picks up from the database are presented to the user.

Click **OK** to save the parameter definition.

By now you have a new dataset and a new report parameter. Next you need to look at the existing data set, which is used to populate the report

Include the LOCATIAONTAG_C column in the report dataset

To modify the FileSystemSet dataset so that the query includes the LOCATIONTAG_C column, complete these steps:

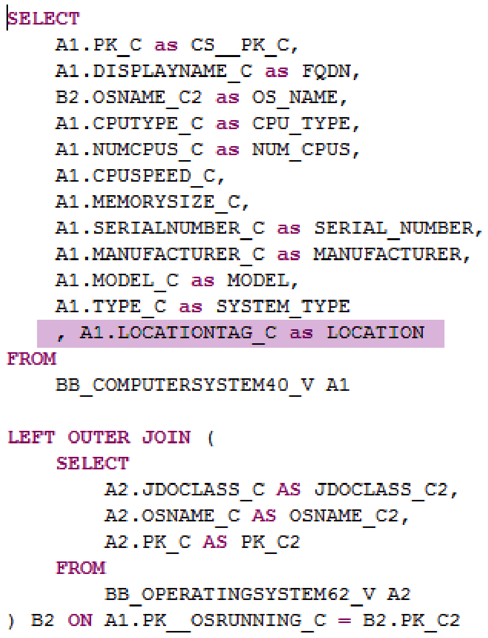
1. From the Data Explorer, explode the Data Sets group, and double-click the data set named FileSystemSet to open it for editing.

2. In the query append the following line immediately before the first FROM clause:

```
, A1.LOCATIONTAG_C AS LOCATION
```

Remember the preceding comma (,).

When you have added the new line, you query should look like this:



```
SELECT
    A1.PK_C as CS__PK_C,
    A1.DISPLAYNAME_C as FQDN,
    B2.OSNAME_C2 as OS_NAME,
    A1.CPUTYPE_C as CPU_TYPE,
    A1.NUMCPUS_C as NUM_CPUS,
    A1.CPUSPEED_C,
    A1.MEMORYSIZE_C,
    A1.SERIALNUMBER_C as SERIAL_NUMBER,
    A1.MANUFACTURER_C as MANUFACTURER,
    A1.MODEL_C as MODEL,
    A1.TYPE_C as SYSTEM_TYPE
    , A1.LOCATIONTAG_C as LOCATION
FROM
    BB_COMPUTERSYSTEM40_V A1

LEFT OUTER JOIN (
    SELECT
        A2.JDOCLASS_C AS JDOCLASS_C2,
        A2.OSNAME_C AS OSNAME_C2,
        A2.PK_C AS PK_C2
    FROM
        BB_OPERATINGSYSTEM62_V A2
) B2 ON A1.PK__OSRUNNING_C = B2.PK_C2
```

3. Click **Preview Results** and verify that the LOCATION column is populated.

   Based on your current data, you may not see actual values for the LOCATION column, however, the important thing is that it is represented in the output. You will most likely have to scroll right to find it.

4. To save your modifications to the FileSystemSet data set, click **OK**.

5. Press **Ctrl+s** to save the entire report.

Adding filtering to include only records for a specific location

Complete these steps to add a filter to the FileSystemSet dataset so that output data are limited to those records that are related to the locations selected by the user,

1. From the Data Explorer, explode the Data Sets group, and double-click the data set named FileSystemSet to open it for editing.

2. In the Filters section, click **New…** to create a new filter.

3. In the New Filter dialog, provide the following values:

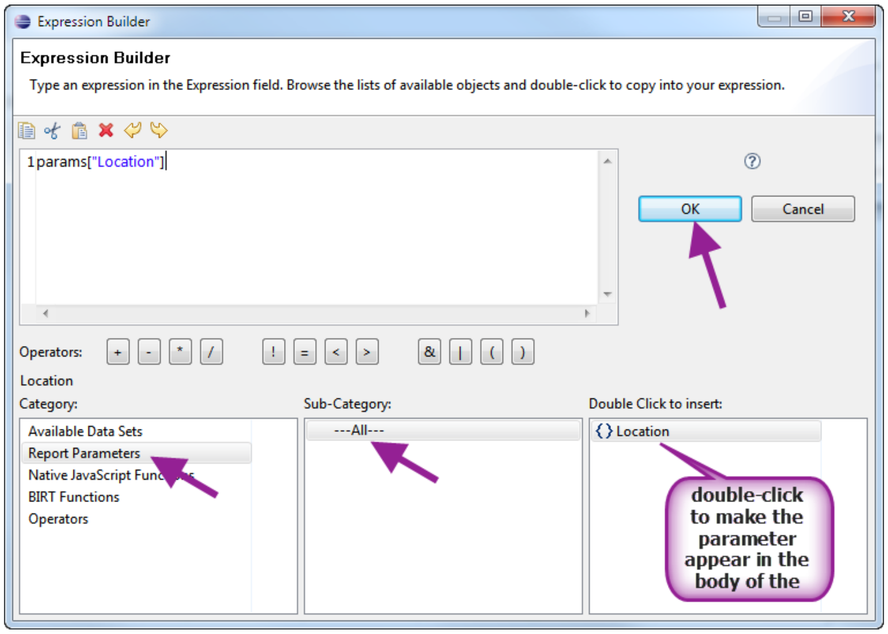
Expression:	`row["LOCATION"]` (select LOCATION from the drop-down list)
Operator:	`In`

   To populate the Value field, select **<Build expression…>** from the drop-down menu of the value field. When the Expression Builder appears, complete these steps:

   a. Select Report Parameters from the Category section.

   b. Choose All from the Sub-Category section.

   c. Double-click Location in the right-most section to insert the parameter reference in the body of the expression.



Expression Builder

Type an expression in the Expression field. Browse the lists of available objects and double-click to copy into your expression.

```
params["Location"]
```

   d. When the expression has been created, click **OK** to save it.

   When you are returned to the New Filter dialog, you will see that the Value field now contains a value of params["Location"].

4. To add the current value for the Value field to the conditions table, click **Add**.

5. Click **OK** to save the filter.

6. Click **OK** in the Edit Data Set dialog to save the dataset.

7. Press **Ctrl+s** to save the entire report.

You have now performed all the necessary modifications to make the report location-aware.

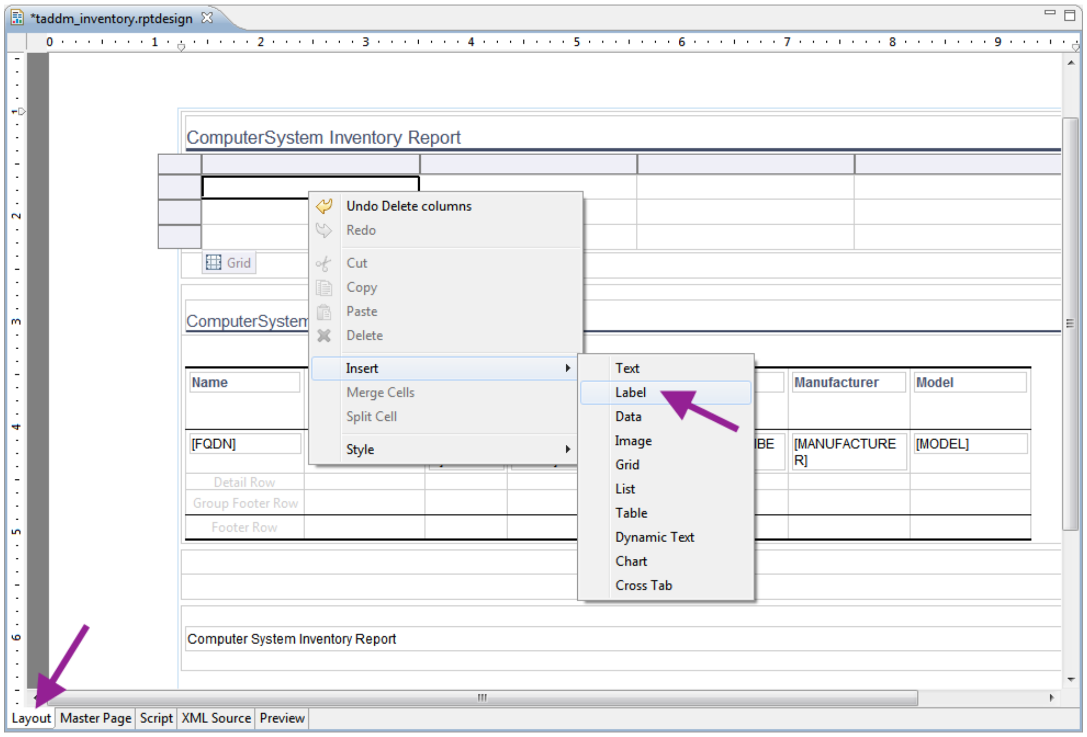
Update the report layout

To display the location information in the report, the report layout must also be changed. Essentially this is not necessary, but you might consider providing the location information in the report so the user can see that their selections were honored.

It is recommended that you insert the value the user has selected in the heading of the report, and add a column to the report body that contains the location of each of the records.

To modify the header, complete these steps:

1. In the Layout pane, right-click on the top-left cell in the header section, and select **Insert** > **Label** from the context menu.



2. Provide a value for the label of `Selected locations`.

3. In the cell, to the right of the new label, right-click and select **Inset** > **Data**.

4. When the New Data Binding dialog appears, provide the following information:

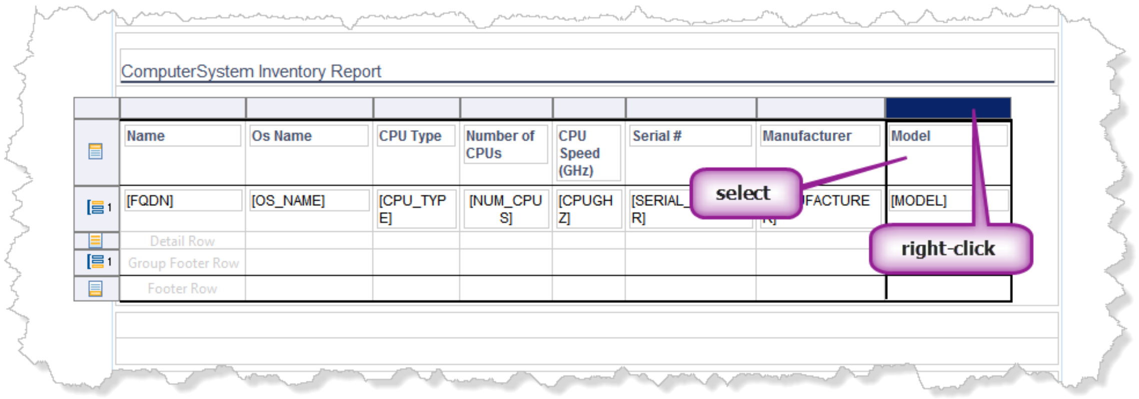
Column Binding Name:	InputLocation
Data Type:	Any
Expression:	params[“Location”] (use the expression builder to select the Location report parameter)

5. To save the new data binding, click **OK**.

You have now ensured that the header of the report will include the location values that was selected by the user.

To add a column to the body of the report, and populate this new column with the location for each record, complete these steps:

1. In the Layout Editor, click on any row in the column named MODEL in the table the represents the body of the report. When you see the table cell and row anchors, right-click on the cell-anchor for the MODEL column, and chose **Insert** > **Column To the Right**.



ComputerSystem Inventory Report

Name	Os Name	CPU Type	Number of CPUs	CPU Speed (GHz)	Serial #	Manufacturer	Model
[FQDN]	[OS_NAME]	[CPU_TYPE]	[NUM_CPUS]	[CPUGHZ]	[SERIAL_R]	FACTURE R]	[MODEL]
Detail Row							
Group Footer Row							
Footer Row							

2. When the new column appears, right-click the top-most column, and choose Insert > Label. When the label text box opens, enter a value of Location.

3. To insert the actual data, right-click the second row of the Location column, and select **Insert** > **Data** from the context menu.

   When the New Data Binding dialog appears, provide the following information:

Column Binding Name:	`LOCATION`
Data Type:	`String`
Expression;	`dataSetRow["LOCATION"]` (consider using the expression builder to populate the field.)

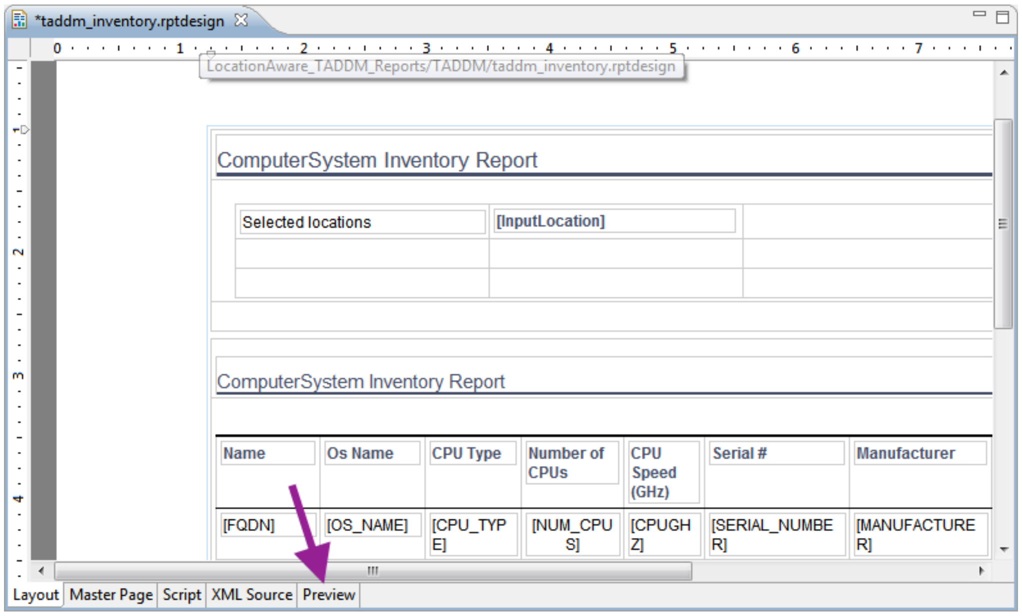
4. To save your modifications, press **Ctrl+s**.

You are done. All the necessary modifications to the standard report has been applied, and you are ready to enjoy the fruits of your labor.

Testing the location-aware report

To test your report, perform these simple steps:

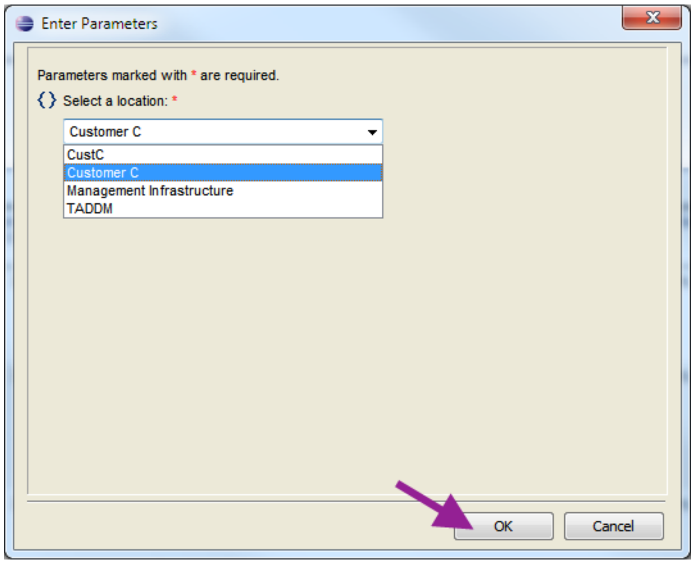
1. Click the Preview tab in the Layout Editor.



2. After a short while, you may be presented with the parameter selection.

   The BIRT Development environment caches your report parameters, so if you have already previewed the report once, the cached parameter values will be used, and the report is displayed immediately. If that is the case, click **Show Report Parameters** to change them.

In the Enter Parameters dialog, notice that only the distinct actual values of the locationTag field in your database are shown. You have most likely a number of resources for which the value of the locationTag attribute is null, and with your current report definitions, there is no way to include these resources in the report. (In the next section you will see how to re-define the report to include null values.)



Parameters marked with * are required.

Select a location: *

Customer C

CustC
Customer C
Management Infrastructure
TADDM

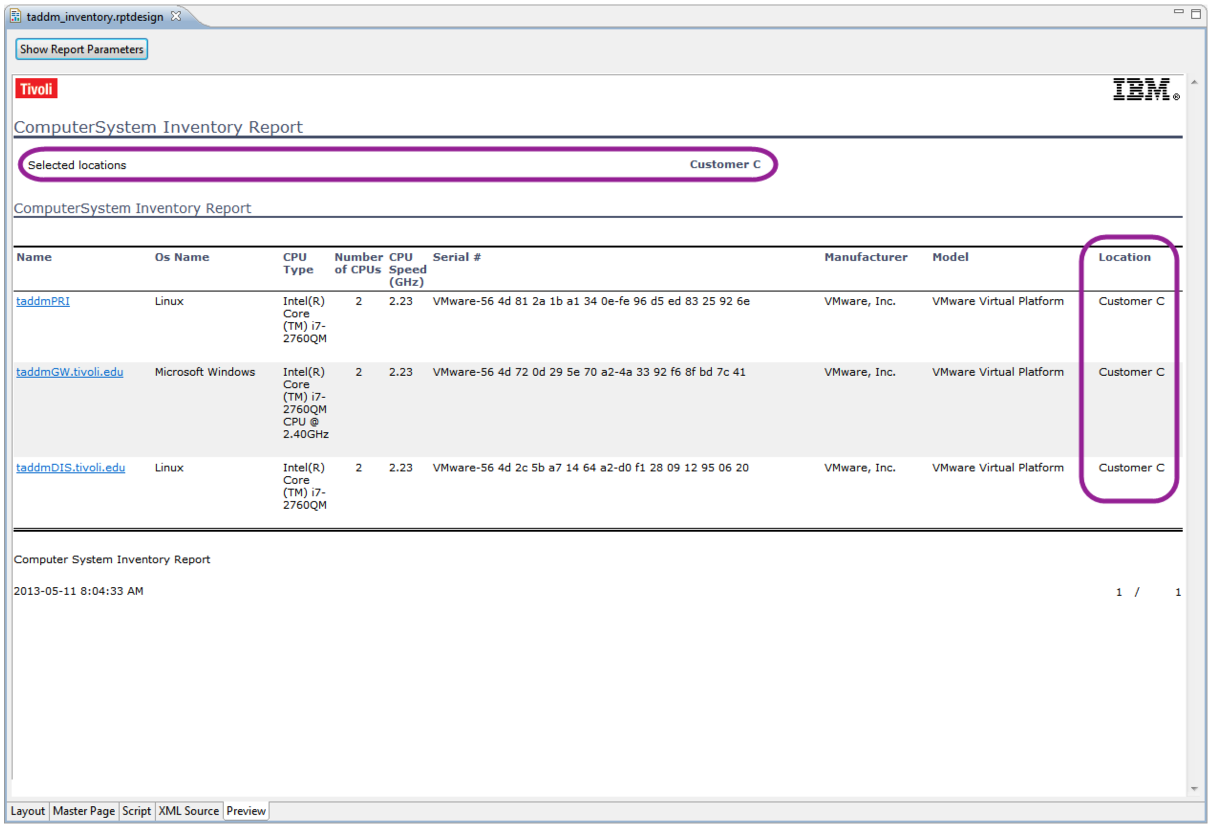
Also notice that only resources from a single location can be included in the report.

To select locations for which you want to report computer system details, simply select the location and click **OK**.

3. When the report is displayed, notice the following:

   - Next to the Selected Locations label in the header section, you see the location you selected from the Enter Parameters dialog.

   - The column named *Location* contains the value of the locationTag attribute for all you resources.



ComputerSystem Inventory Report

Selected locations Customer C

ComputerSystem Inventory Report

Name	Os Name	CPU Type	Number of CPUs	CPU Speed (GHz)	Serial #	Manufacturer	Model	Location
taddmPRI	Linux	Intel(R) Core (TM) i7-2760QM	2	2.23	VMware-56 4d 81 2a 1b a1 34 0e-fe 96 d5 ed 83 25 92 6e	VMware, Inc.	VMware Virtual Platform	Customer C
taddmGW.tivoli.edu	Microsoft Windows	Intel(R) Core (TM) i7-2760QM CPU @ 2.40GHz	2	2.23	VMware-56 4d 72 0d 29 5e 70 a2-4a 33 92 f6 8f bd 7c 41	VMware, Inc.	VMware Virtual Platform	Customer C
taddmDIS.tivoli.edu	Linux	Intel(R) Core (TM) i7-2760QM	2	2.23	VMware-56 4d 2c 5b a7 14 64 a2-d0 f1 28 09 12 95 06 20	VMware, Inc.	VMware Virtual Platform	Customer C

Computer System Inventory Report

2013-05-11 8:04:33 AM 1 / 1

4. If you want to test the report with different data, click **Show Report Parameters**, and select a different location.

As you have seen, the modifications you need to make to the standard reports are very simple, and straightforward to implement, and if you decide not to include the location information in the report, it is even easier. In a multi-customer environment you can use the locationTag (or any other customer-specific attribute) to differentiate between each of the organizations you support, and use the techniques described above to produce reports that are specific to each customer.

The modifications you applied to the tivoli_inventory.rptdesign were very basic, and can be used if you want reports for one location at a time. But what if you need to include multiple locations, and also need to be able to include resources for which the locationTag attribute has not been populated?

The next section demonstrates how to refine your report definitions to overcome these restrictions.

5.2.2.2 Advanced location-aware report design

To provide greater flexibility in the report you should refine the simple location-aware report so it allow the user to select resources from multiple locations, including any combination of specific locations and *all locations*(locationTag has any value or is null), *any location* (locationTag has a value), or *no location* (locationTag is null).

To facilitate this you must apply the following modifications:

- Augment the Locations dataset to include selections for All Locations, Any Location, and No Location.
- Show the number of records for each location in the parameter selection dialog.
- Enable multi-value selection for the Location report parameter.
- Modify the query in the FileSystemSet data set to use a static where clause
- Provide a script in the beforeOpen event for the FileSystemSet dataset that reads the report parameter and supplies the appropriate where clause.
- Modify the data binding for the display of the report parameter in the report header in order to display all selected input values.

In the following, the necessary steps need to apply each of these modifications are described.

Augment the Locations dataset to include additional values.

To enable the user to select values for the input parameter that is not represented in the database, you can take one of two approaches.

- Create two new data sets and (one that contains your dynamic selection values, and one that joins your new data set with the locations found in the database) and link the new join dataset to the report parameter.
- Modify the existing query in the Locations dataset to include the dynamic values.

In this case you will modify the query in the Locations dataset.

Complete these steps to force the Locations dataset query to return values that represent all locations as well as *no location*, *any location*, and *all locations* and instruct it to return the row-count for each location:

1. From the Data Explorer, open the Locations dataset, and modify the query so it looks like this:

```
select
      CASE
      WHEN LOCATIONTAG_C is NULL THEN 'no location'
      ELSE LOCATIONTAG_C
      END
      as Location,
      count(*) as Count
from BB_COMPUTERSYSTEM40_V
group by LOCATIONTAG_C

union

-- include 'all locations'
select 'all locations' as Location, count(*) as Count from
BB_COMPUTERSYSTEM40_V L

union

-- include 'any locations' if the database contains records with locationTag
!= NULL
select 'any location' as Location, count(*) as Count
from BB_COMPUTERSYSTEM40_V
where LOCATIONTAG_C is not NULL
```

The first section of the query selects all know locations and substitutes a value of NULL with *no location*. In addition, it returns to row-count for each location.

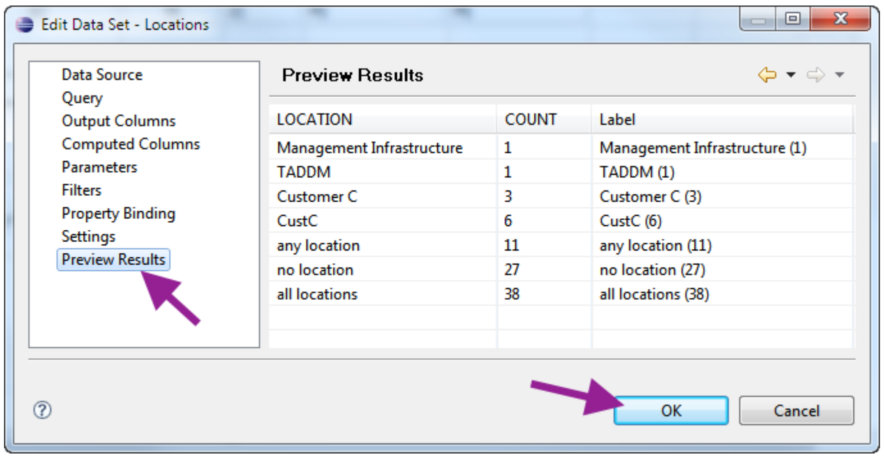
The second section adds *all locations* and the related count to the output set, and the third adds *any location* and the count of all records where the locationTag attribute has a value.

By returning both the location and the count, you can create a label that includes both location and count, and display that to the end user.

a. To create the label, click **Computed Columns**, and add a new column with the following specification:

Column name:	`Label`
Data Type:	`String`
Expression:	`row["LOCATION"] + " (" + row["COUNT"] + ")"` use the expression builder to include the two rows from the dataset.

b. To verify your results, click **Preview Results**, and verify that your output is similar to this example:



LOCATION	COUNT	Label
Management Infrastructure	1	Management Infrastructure (1)
TADDM	1	TADDM (1)
Customer C	3	Customer C (3)
CustC	6	CustC (6)
any location	11	any location (11)
no location	27	no location (27)
all locations	38	all locations (38)

Notice how the Label field contains a user friendly string that contains both the location and the record count.

To save your updates, click **OK**.

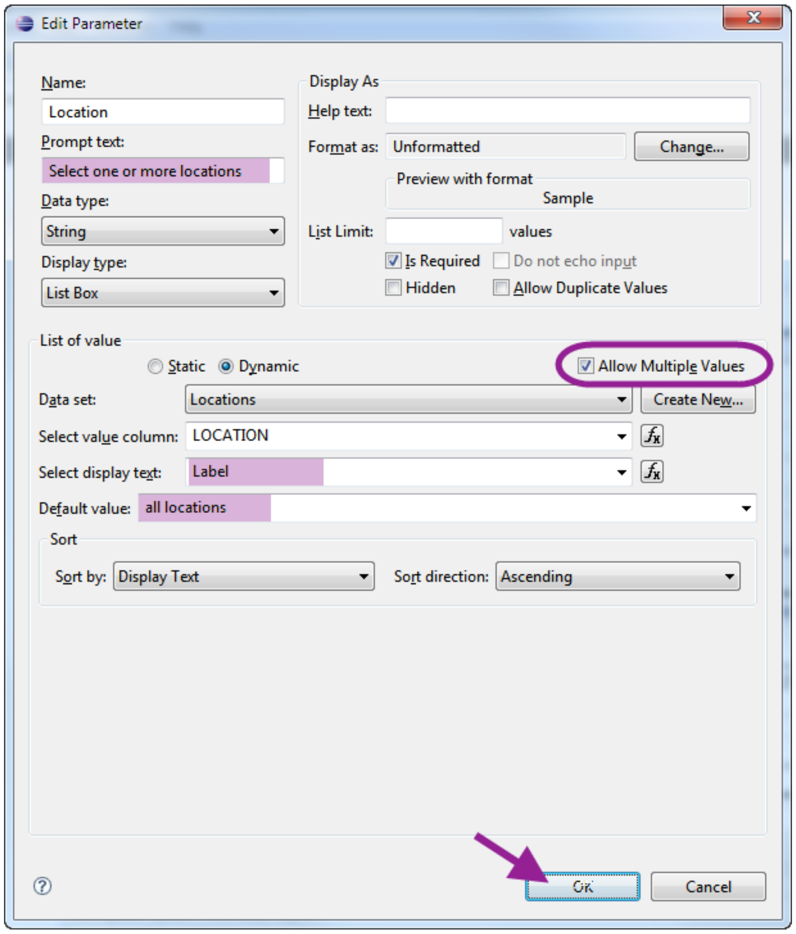
You have now prepared the data set to provide additional values and user friendly label.

2. Next you need to modify the report parameter to use the user friendly label, and allow multi-value selection. To implement this, complete these steps:

a. From the Data Explorer, open the Location report parameter, and change the properties in accordance with the following specifications:

Prompt text:	`Select one or more locations`
Allow Multiple Values:	`checked`
Select display text:	`Location`
Default value:	`all locations`

When you are done, the Edit report Parameter should look like this:



Name: Location
Prompt text: Select one or more locations
Data type: String
Display type: List Box

Display As
Help text:
Format as: Unformatted
Preview with format: Sample
List Limit: values
[✓] Is Required [ ] Do not echo input
[ ] Hidden [ ] Allow Duplicate Values

List of value
Static [✓ Dynamic] [✓] Allow Multiple Values
Data set: Locations
Select value column: LOCATION
Select display text: Label
Default value: all locations
Sort
Sort by: Display Text
Sort direction: Ascending

b. Click **OK** to save your work.

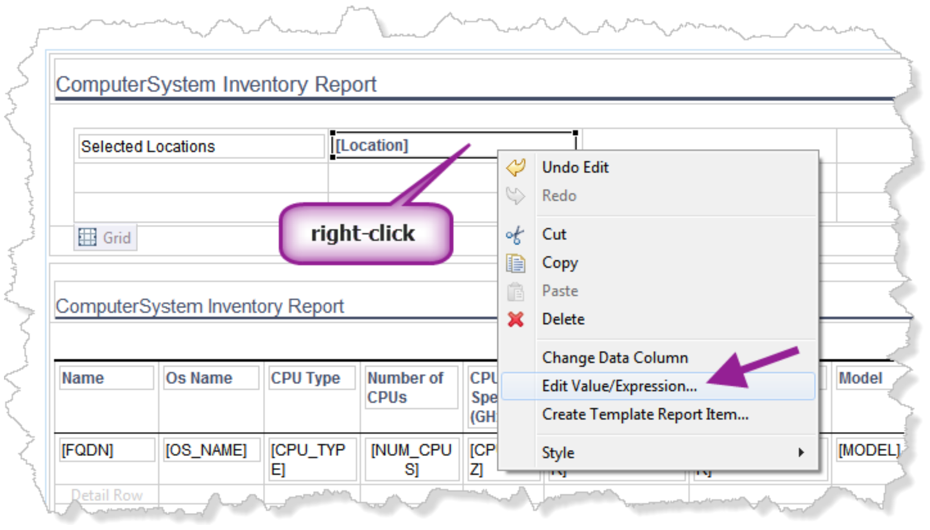
At this point, you have changed the report parameter so that the user will see the number of records in each location, and can select multiple locations to be included in the report.

3. Parse array report parameter

Unfortunately, because the parameter now contains multiple values, it will not be displayed correctly in the Location field you added to the report header. In order to fix this, you must modify the data binding for the field.

Complete these steps:

1. From the Layout Editor, right-click the Location field in the report header section, and choose **Edit Value/Expression**.



a. When the Edit Data Binding dialog appears, use the expression builder tool (*fx*) to modify the Expression field.

b. In the body of the expression, insert the following script.

```
strParamValsSelected=reportContext.getParameterValue("Location")

var strSelValues=""

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

{ strSelValues += strParamValsSelected[i].toString()+" ,"}

strSelValues = strSelValues.substring(0,strSelValues.length-2);
```

This script parses each member of the Location report parameter as a string, and separates them values with commas

c. To save the new binding, click **OK** in the expression builder, and **OK** in the Edit Data Binding dialog.

d. To save the entire report, press **Ctrl+s**.

By now, you are almost ready. You have applied all the cosmetic modifications that will make the query more static, all that is left is to modify the FileSystemSet data set so the correct data are presented in the report.

Applying a script to provide where clause

The final modifications you need to apply will redefine the query in the FileSystemSet dataset so the where clause in the query is populated from a script. The script is added in the beforeOpen event of the dataset, and uses the selected parameters to build an appropriate where clause based on the content of the Location report parameter.

To change the query so it receives a static where clause, all you do is to add a placeholder, which then will be updated by the script. To add this placeholder, complete these steps:

1. From the Data Explorer, open the FileSystemSet dataset, and look at the query. Notice that the query does not contain a where clause. This implies that all filtering is performed by the filters associated with the dataset.

2. Append the following where clause at the end of the query:

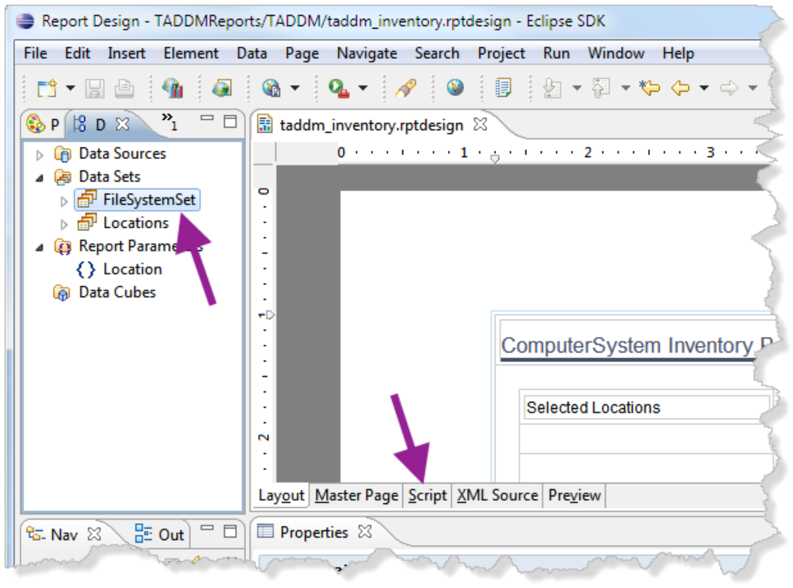
   ```
   where *WHERE-CLAUSE_FROM_SCRIPT*
   ```

   This *where* clause does not adhere to the specifications in the SQL language. For that reason, you will no longer be able to preview the results, because you, deliberately, have provided an invalid query. However, in the next step you provide a script that replaces the invalid syntax at runtime, so that the query actually will produce data.

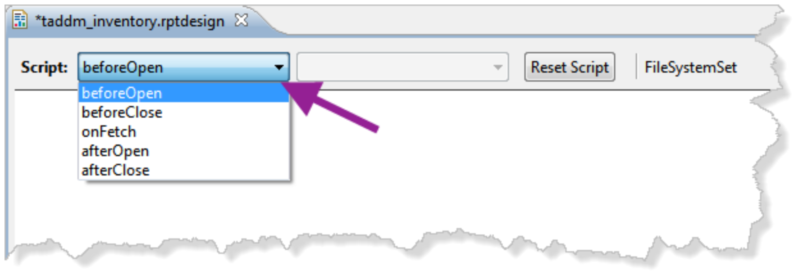
3. Click **OK** to save the dataset.

4. To create the script that will be responsible for inserting a valid where clause instead of the **WHERE-CLAUSE_PLACE_HOLDER** string in the query, complete these steps:

   a. From the Data Explorer, select (double click) the FileSystemSet dataset, and open the Script tab in the Layout Editor.


Report Design - TADDMReports/TADDM/taddm_inventory.rptdesign - Eclipse SDK
File Edit Insert Element Data Page Navigate Search Project Run Window Help
taddm_inventory.rptdesign
Data Sources
Data Sets
FileSystemSet
Locations
Report Parame
Location
Data Cubes
ComputerSystem Inventory
Selected Locations
Layout
Master Page
Script
XML Source
Preview
Nav
Out
Properties

b. In the Script Editor, ensure that you have selected beforeOpen as the event for which you want to provide a script.



c. Insert the following script in the body of the beforeOpen script.

```
// receive the report parameter named Location
var strSelValues="";
strParamValsSelected=reportContext.getParameterValue("Location");
for (var i=0;i<strParamValsSelected.length;i++)
{
    strSelValues += strParamValsSelected[i].toString()+",";
}
strSelValues = strSelValues.substring(0,strSelValues.length-1);

// build and array of locations from the report parameter
var selectedValues = strSelValues.split(",")

// initialize
var process_all = false
var process_any_location = false
var process_no_location = false
var process_location = false
var locationList = ""

// determine what to do
for ( var i=0;i<selectedValues.length;i++) {

   if ( selectedValues[i] == "all locations" ) {
          process_all = true;
   } else if ( selectedValues[i] == "any location" ) {
          process_any_location = true;
   } else if ( selectedValues[i] == "no location" ) {
          process_no_location = true;
   } else {
          process_location = true
          if (locationList != "") {
                 locationList = locationList + ","
          }
          locationList = locationList + "'" + selectedValues[i] + "'"
   }
}

// create the appropriate where clause
var where = ""

if (process_all) {               // all locations
   where = " 1 = 1";
} else if ( (process_any_location) && (process_no_location) ) {       // any locations and all
empty
   where = " 1 = 1 "
} else if ( (process_any_location) && !(process_no_location) ) {      // any location
   where = "LOCATIONTAG_C IS NOT NULL"
} else if ( (process_no_location) && (process_location) ) {           //has specific or no
location
   where = "LOCATIONTAG_C IS NULL OR LOCATIONTAG_C IN (" + locationList + ")"
} else if (process_no_location)  {              // has no location
   where = "LOCATIONTAG_C IS NULL"
} else {                                                // has specific location
   where = "LOCATIONTAG_C IN ("+ locationList + ")"
}

// update the query
this.queryText = this.queryText.replace("*WHERE-CLAUSE_FROM_SCRIPT*",where);
```

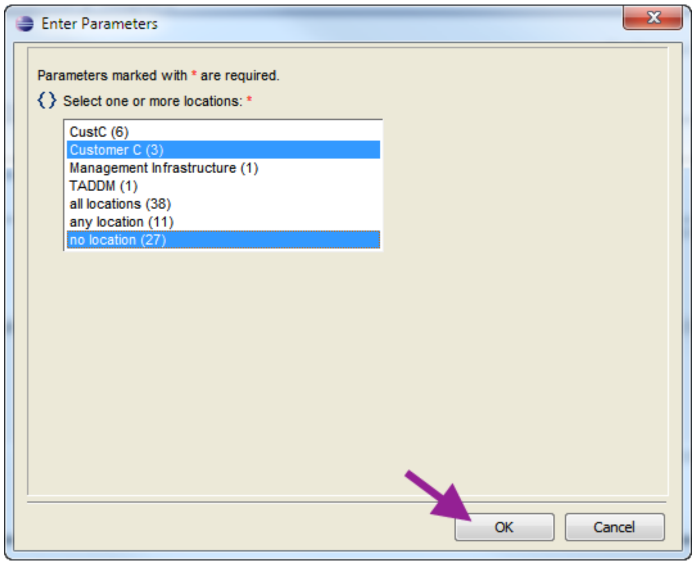
d. Click **Ctrl+s** to save the report.

You have now performed all the modifications necessary to make the report location-aware, and to provide user-friendly parameter selection facilities, so any combination of locations can be selected.

Testing the advanced location-aware report

To test the report, simply open the Preview tab in the layout editor. Because you already tested the report once, the report is immediately opened using the parameter selection from the previous run.

To see the new parameter dialog, click **Show Report Parameters**, and notice which options are available in the *Select one or more locations* section:



Enter Parameters

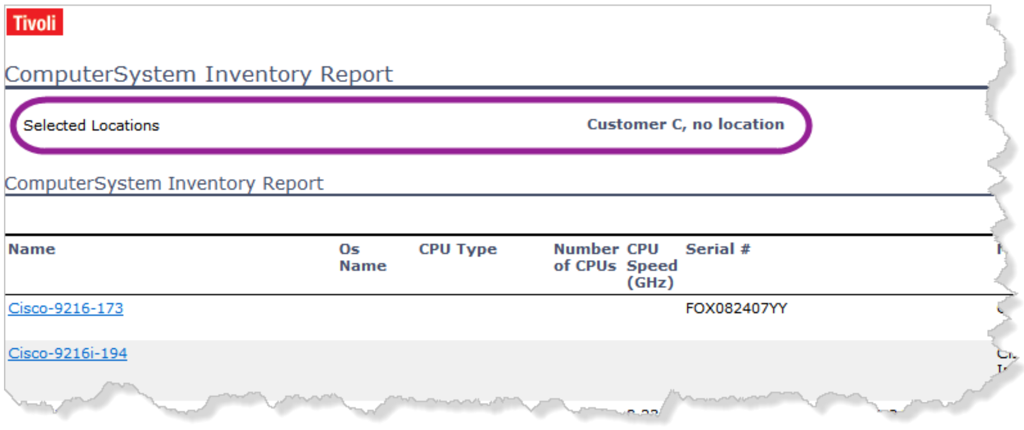
Parameters marked with * are required.

Select one or more locations: *

- CustC (6)
- Customer C (3)
- Management Infrastructure (1)
- TADDM (1)
- all locations (38)
- any location (11)
- no location (27)

Notice how the record counts are shown in parenthesis next to each location, and that you can select multiple locations. At the bottom you see the three new placeholder locations: *all locations*, *any location*, and *no location*.

To create a new report, select any combination of locations you like, and click **OK.**



ComputerSystem Inventory Report

Selected Locations — Customer C, no location

ComputerSystem Inventory Report

Name	Os Name	CPU Type	Number of CPUs	CPU Speed (GHz)	Serial #
Cisco-9216-173					FOX082407YY
Cisco-9216i-194					

Notice how all the locations you selected are displayed in the header of the report.

This completes the examples on making your BIRT reports location-aware.

5.2.3 Cognos Reports and location tagging

As stated in the TADDM documentation, no out-of-the-box reports have been modified to support the use of the locationTag attribute. The good news is that all the building-block views have been updated to support the attribute.

You saw in the previous sections, that because the views include the locationTag attribute, all the necessary modifications in the BIRT environment are related to the individual reports, their data, layout and use of parameters. If you use Cognos for reporting, you are faced with similar challenges. However, because Cognos uses a logical representation of the data in the TADDM database, known in Cognos as a *model*, you must also update the TADDM model to include the new fields from the building-block views.

5.2.3.1 Updating the TADDM model

The Cognos TADDM model is updated using the windows-based Cognos Framework Manager tool. In the following it is assumed that you have access to an operational Tivoli Common Reporting (TCR) environment and that you have imported the TADDM package and configured the *CMDBTCR* data source. It is also assumed that you have installed Cognos Framework Manager, and configured it to connect to your TCR server,

To modify the TADDM model, you must perform these tasks:

- Create a new Cognos Framework Manager project and import the most current TADDM model.
- Update the query subjects by adding new query items that represent the locationTag attribute.
- Publish the modified model to the TCR server.

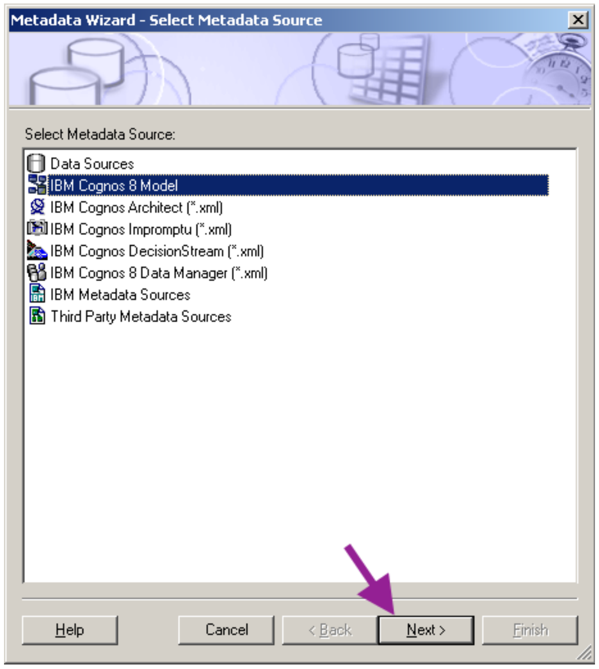
Create the TADDM project and import the TADDM model

Before you start this task, ensure that you, from the system where you are running the Framework Manager tool, have access to the `model.xml` file from the `$COLLATION_HOME/dist/etc/reporting/tcr` directory on your TADDM server. If necessary, copy the `model.xml` file to your windows system before starting this task.

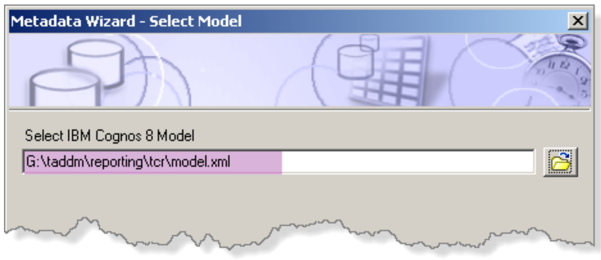
Complete these steps to create a new Cognos Framework Manager project:

1. Open the Cognos Framework Manager tool, and create a new project named `TADDM`.

2. When the *Metadata Wizard - Select Metadata Source* dialog appears, select a source type of **IBM Cognos 8 Model**. Click **Next** when you are ready.


Metadata Wizard - Select Metadata Source
Select Metadata Source:
Data Sources
IBM Cognos 8 Model
IBM Cognos Architect (*.xml)
IBM Cognos Impromptu (*.xml)
IBM Cognos DecisionStream (*.xml)
IBM Cognos 8 Data Manager (*.xml)
IBM Metadata Sources
Third Party Metadata Sources
Help
Cancel
< Back
Next >
Finish

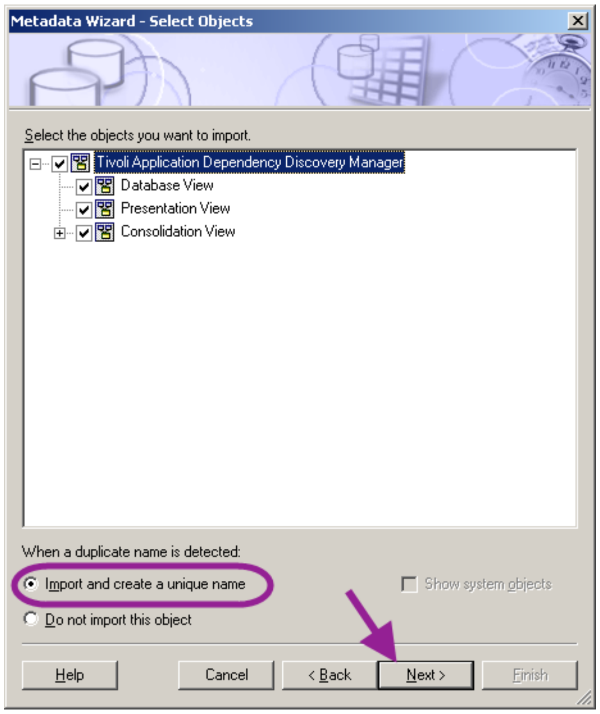
3. In the *Metadata Wizard - Select Model* dialog, choose the model.xml file you made available to the system hosting the Framework Manager tool, and click **Next**.



Select IBM Cognos 8 Model

G:\taddm\reporting\tcr\model.xml

4. Finally, the *Metadata Wizard - Select Objects* dialog, select all the available objects, ensure that the *Import and create a unique name* option is selected, and click **Next**.


Metadata Wizard - Select Objects
Select the objects you want to import.
Tivoli Application Dependency Discovery Manager
Database View
Presentation View
Consolidation View
When a duplicate name is detected:
Import and create a unique name
Do not import this object
Show system objects
Help
Cancel
< Back
Next >
Finish

5. When you see that the import completed, click **Finish**.

By now you have imported the TADDM data model into the Cognos Framework Manager tool. You are now ready to apply your modifications.

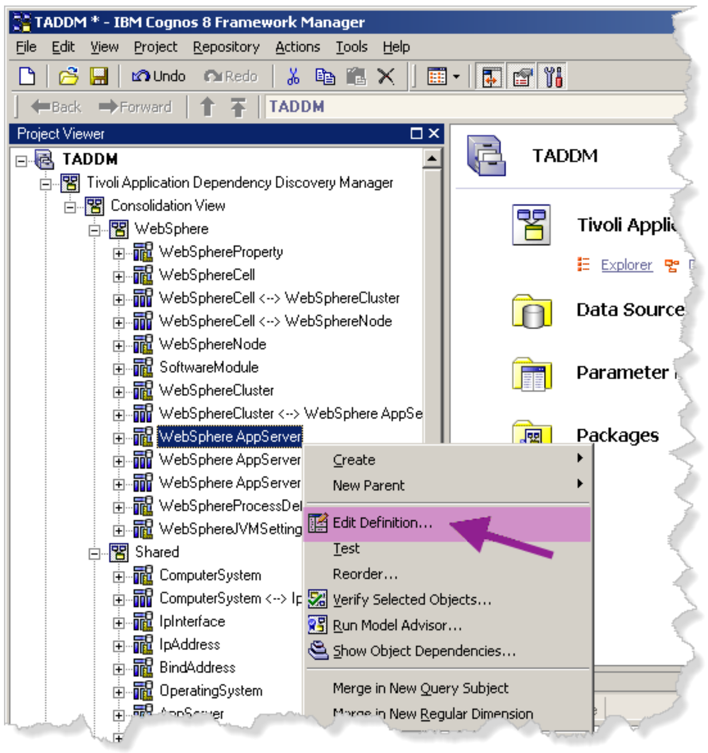
Modify queries to include the locationTag attribute

The TADDM model contains 49 query subjects, each of which selects data from the building-block views in the TADDM database. In order to include the locationTag in your reports, you must update the specific query subjects that are referenced by the reports. You can choose to update only the query subjects for which you anticipate that the locationTag will be used (for example ComputerSystems, and AppServers) or all of the query subjects, to be prepared for future requirements.

In the following example, the WebSphere AppServer query subject is modified so the locationTag attribute is included in the query subject as a new query item.

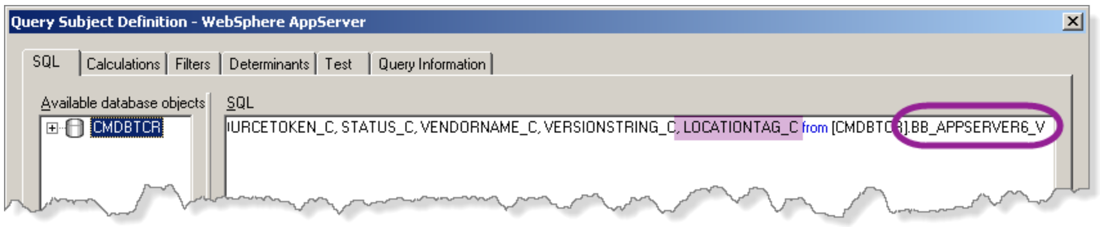
To add the LOCATIONTAG_C column from a building-block view to the Cognos query subject, complete these steps:

1. From the Framework Manager Project Viewer, explode the hierarchy, and select the query subject you wish to update.



Right-click the query subject, and choose **Edit Definition** from the context menu.

2. In the SQL tab of the Query Subject Definition dialog you see a long, single-line SQL select statement. If you scroll to the far right, you will see the name of the building-block view that data are selected from.


Query Subject Definition - WebSphere AppServer
SQL
Calculations
Filters
Determinants
Test
Query Information
Available database objects
CMDBTCR
SQL

```
URCETOKEN_C, STATUS_C, VENDORNAME_C, VERSIONSTRING_C, LOCATIONTAG_C from [CMDBTCR].BB_APPSERVER6_V
```

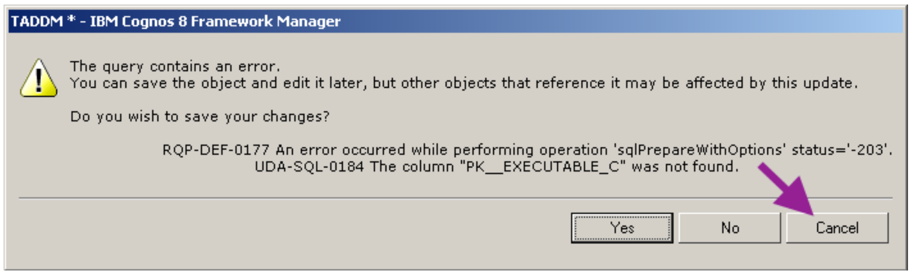
To include the LOCATIONTAG_C column, insert the string

```
, LOCATIONTAG_C
```

Immediately before the *from* clause. Remember to include the preceding comma, and leave a space before *from*.

3. Click **OK** to save your modifications.

   At this point the SQL select statement is validated. You may see validation errors because the building-block views no longer contain the all referenced columns.



TADDM * - IBM Cognos 8 Framework Manager

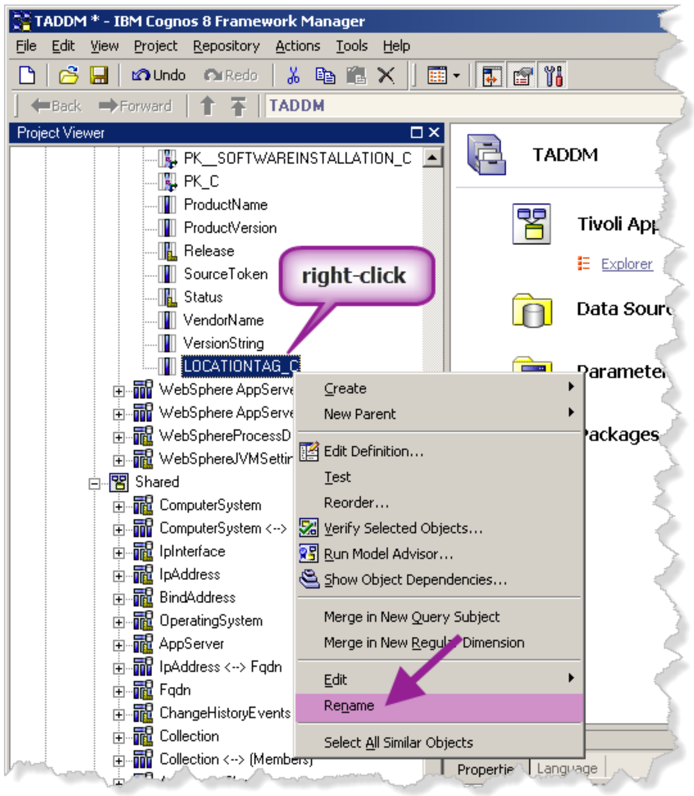
The query contains an error.
You can save the object and edit it later, but other objects that reference it may be affected by this update.

Do you wish to save your changes?

RQP-DEF-0177 An error occurred while performing operation 'sqlPrepareWithOptions' status='-203'.
UDA-SQL-0184 The column "PK__EXECUTABLE_C" was not found.

   If you experience such validation errors, click **Cancel**, remove the violating column references in the select statement, and try to save the query subject again.

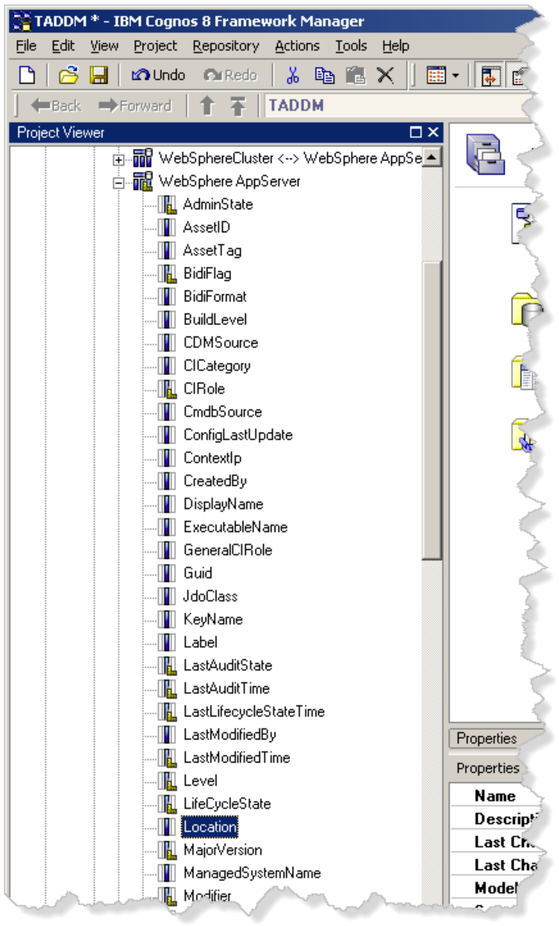
4. Once the query subject has been saved, you will notice that the LOCATIONTAG_C query item has been added as the last item in the query subject. To rename it so it appears to the report designers as *Location*, simply right-click the query item, select **Rename** from the context menu, and provide a new name of `Location`.



5. You can consider moving the Location query item to a more appropriate location in the query set. The items already appear in alphabetic order, so you may want to drag the item to the location where it would naturally fit.

Note: You could have avoided this move step if you had inserted the reference to the LOCATIONTAG_C column in the SQL select statement at the proper location in the select sequence.

In this example, the *Location* query item should logically appear after the *LifeCycleState* item, as shown below:



6. To save the model, press **Ctrl+s**.

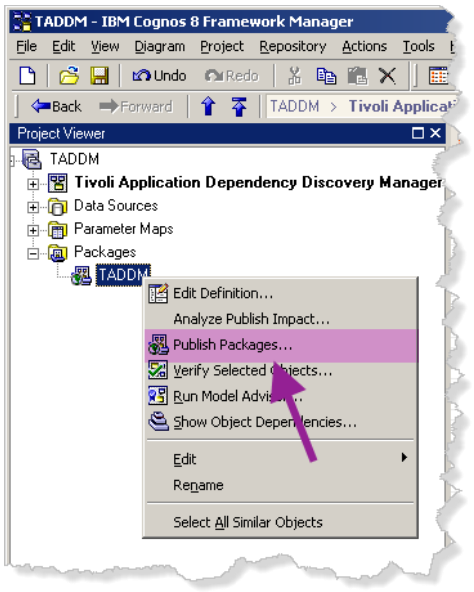
7. Consider updating other query subjects for which you anticipate that the locationTag attribute will be included.

You are done. The model has been updated to include the locationTag attribute. All that is needed before you can reference it in the Cognos reports is to build a new version of the TADDM package and export it to the TCR server.

Publish the modified model to the TCR server

To make the modified version of the TADDM model available in the TCR environment, all you need to do is to the TADM package to the TCR server. The following steps demonstrate how to achieve this:

1. In the Cognos Framework Manager Project View open the *Packages* section, right-click the TADDM package, and select **Publish Packages** from the context menu.


TADDM - IBM Cognos 8 Framework Manager
File Edit View Diagram Project Repository Actions Tools
Undo Redo
Back Forward TADDM > Tivoli Applicat
Project Viewer
TADDM
Tivoli Application Dependency Discovery Manager
Data Sources
Parameter Maps
Packages
TADDM
Edit Definition...
Analyze Publish Impact...
Publish Packages...
Verify Selected Objects...
Run Model Advisor...
Show Object Dependencies...
Edit
Rename
Select All Similar Objects

2. In the *Publish Wizard - Select Location Type* dialog, use the default selection (IBM Cognos 8 Content Store), and click **Next**.

Note: In this example, versioning are not being used. If you want to be able to use several versions of the package in your TCR environment, consult your local TCR/Cognos administrator.

3. Unless you have specific requirements, accept the default settings in the *Publish Wizard - Add Security* dialog, and click **Next**.

4. In the *Publish Wizard - Options* dialog, click **Publish** to upload the package to the TCR server.

   At this point you will receive a warning indicating that you are about to overwrite an existing package. To accept this, press **Yes**.

5. When the upload completes, press **Finish** to dismiss the Publish Wizard.

Your job is done. You can now inform the report developers that the model has been augmented with the location information, so they can get started developing location-aware reports.

You may consider exporting the new model to your local file system so you can easily apply the same modifications to another environment, or compare your model with the model delivered with the next version of TADDM.

5.2.3.2 Including location in a report

Once the new model has been uploaded to the TCR server, you can add the location information to your existing reports. The following example outlines how to achieve that for the *sample-websphere.applications* report.

To modify the report, complete these steps:

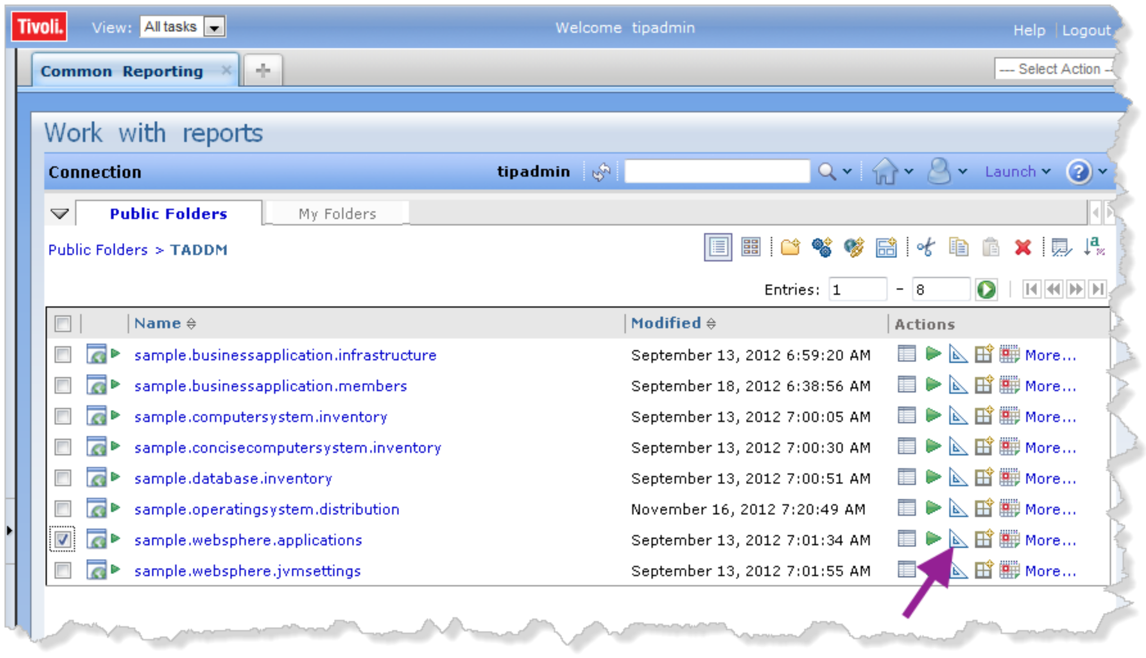
1. To load the report into the Report Studio, complete these steps:

   a. Log in to the Tivoli Integrated Portal hosting your TCR server.

   b. Open the **Reporting** > **Common Reporting** application.

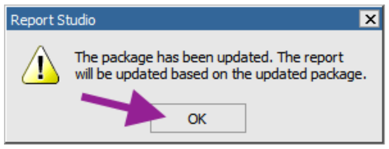
   c. Open the **TADDM** folder.

   d. Load the sample .websphere.applications report in the Report Studio by clicking the Open with Report Studio icon(◺).



Name	Modified	Actions
sample.businessapplication.infrastructure	September 13, 2012 6:59:20 AM	More...
sample.businessapplication.members	September 18, 2012 6:38:56 AM	More...
sample.computersystem.inventory	September 13, 2012 7:00:05 AM	More...
sample.concisecomputersystem.inventory	September 13, 2012 7:00:30 AM	More...
sample.database.inventory	September 13, 2012 7:00:51 AM	More...
sample.operatingsystem.distribution	November 16, 2012 7:20:49 AM	More...
sample.websphere.applications	September 13, 2012 7:01:34 AM	More...
sample.websphere.jvmsettings	September 13, 2012 7:01:55 AM	More...

2. When the Report Studiio opens, you may receive a warning message indicating that the package has been updated. Hopefully you understand why.

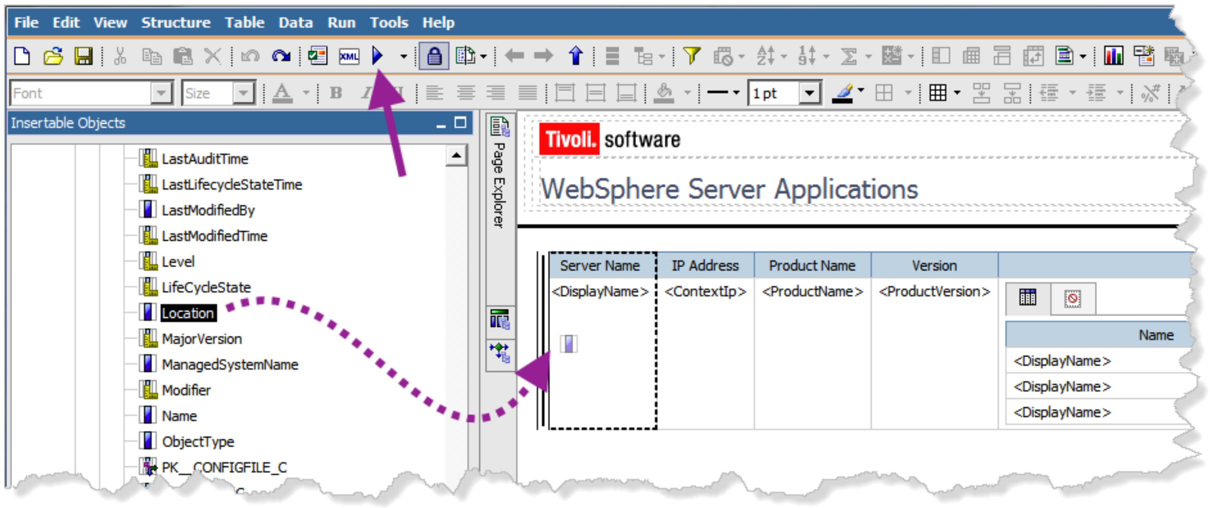


The package has been updated. The report will be updated based on the updated package.

Click **OK** to dismiss the message.

3. In the Insertable Objects pane, explode the path **TADDM** > **Consolidation View** > **WebSphere** > **WebSphere AppServer**, and locate the item named *Location*.

4. To include the Location information as the first column in the report body, drag the Location item to the Layout Editor, and drop it on the left-most border.



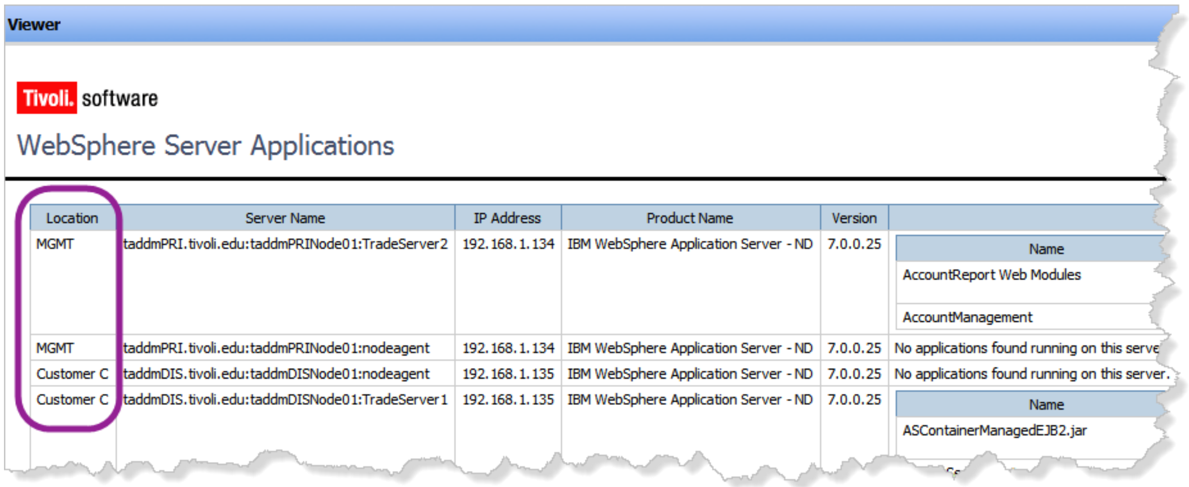
Server Name	IP Address	Product Name	Version
<DisplayName>	<ContextIp>	<ProductName>	<ProductVersion>

Name
<DisplayName>
<DisplayName>
<DisplayName>

When you drop the item, notice that a new column is inserted in the report.

5. To test the report, click the Run tool (▶)in the menu bar.

After a short while, you will see the new report that includes the location. Depending on your data, it may look similar to the following example::



WebSphere Server Applications

Location	Server Name	IP Address	Product Name	Version	
MGMT	taddmPRI.tivoli.edu:taddmPRINode01:TradeServer2	192.168.1.134	IBM WebSphere Application Server - ND	7.0.0.25	Name: AccountReport Web Modules; AccountManagement
MGMT	taddmPRI.tivoli.edu:taddmPRINode01:nodeagent	192.168.1.134	IBM WebSphere Application Server - ND	7.0.0.25	No applications found running on this serve
Customer C	taddmDIS.tivoli.edu:taddmDISNode01:nodeagent	192.168.1.135	IBM WebSphere Application Server - ND	7.0.0.25	No applications found running on this server.
Customer C	taddmDIS.tivoli.edu:taddmDISNode01:TradeServer1	192.168.1.135	IBM WebSphere Application Server - ND	7.0.0.25	Name: ASContainerManagedEJB2.jar

Naturally you can apply additional modifications to the Cognos reports to support parameter selection , and filtering on specific locations. The way this is impplemented in Cognos is very similar to the way it works in BIRT, so let yourself be inspired by the examples in *5.2.2.2 Advanced location-aware report design* on page 41.

This completes the section on including location information in your Cognos reports.

5.3 Visualizing the location in the Data Management Portal

If you wish to visualize the locationTag in the Detail View pane of the Data Management Portal, you must customize the views that control the content of the Detail Views.

The logical location for visualizing the locationTag is in the CI Info tab, which contains common lifecycle related information such as Asset Tag, CI Category, Audit State, and Life Cycle State. Of course you can choose to show the locationTag information in the General tab alongside resource name. However, if you choose this path, you must customize the views for each specific resource type for which you want to include the locationTag.

Note: If you customize the content or layout of the details panes in the TADDM Data Management Portal, you should be aware of the fact that your customizations will be overwritten when you apply a fixpack or upgrade to the next level of TADDM.

It is recommended that you save your custoized xml files in a location that is not overwritten by the TADDM installation/upgrade procedure, so you can reapply your changes.

Consider adding the files to a custom server template and discover your TADDM servers so you keep track of the change history of yor customizations.

Customization of the Detail Pane views is performed by modifying the following files:

- `attributenames_<locale>.xml`
- `screencontent_<locale>.xml`

Both files reside in the `$COLLATION_HOME/etc/detail` directory. If multiple TADDM Storage Servers are used in your environment, you must manually synchronize the content of the files across all your TADDM Storage Servers. (Consider using TADDM to discover the Storage Servers so you apply change control to these files. This will allow you to compare and control changes to them).

The `screencontent_<locale>.xml` file specifies what information to display in which tabs in the Detail Pane for each top-level resource type. The `attributenames_<locale>.xml` file contains the mapping between the attributes and the labels that are displayed.

To customize the CI Info tab to include the locationTag attribute, complete these steps:

1. On the TADDM Storage Server(s) that support the Data Management Portal apply these modifications to the `$COLLATION_HOME/etc/detail/screencontent_<locale>.xml` file:

   a. Copy the file `screencontent_<locale>.xml` to `screencontent_<locale>.xml.original` to keep a backup copy.

   b. Open the `screencontent_<locale>.xml` file in your favorite editor

   c. Locate the section that starts with:

   ```
   <!-- ConfigurationItem START -->
   ```

   d. Scroll down a few lines, and locate the line that defines the attribute you want to precede the locationTag attribute. Insert the following line after the line you identified:

   ```
   <plain fieldName="locationTag"/>
   ```

   e. Add any additional attributes you want to include, for example *CIRole* or *generalCIRole.*

   f. Save the file.

   If you restart the TADDM Storage Server at this point, you will see that the attributes now appear in the CI Info tab in the Details pane. However the label(s) that are displayed for your new attributes will show *null*.

2. To provide the correct label, you must update the `attributenames_<locale>.xml` file. Complete these steps to apply valid label(s) for the attribute(s) you added.

   a. Copy the file `attributenames _<locale>.xml` to `attributenames_<locale>.xml.original` to keep a backup copy.

   b. Open the `attributenames _<locale>.xml` file in your favorite editor

c. Locate the line that contains:

```
<attribute
className="com.collation.platform.model.topology.process.itil.Configura
tionItem" displayName="Asset ID" name="assetID"/>
```

d. Insert the following line after the line you identified:

```
<attribute
className="com.collation.platform.model.topology.process.itil.Configura
tionItem" displayName="Location" name="locationTag"/>
```

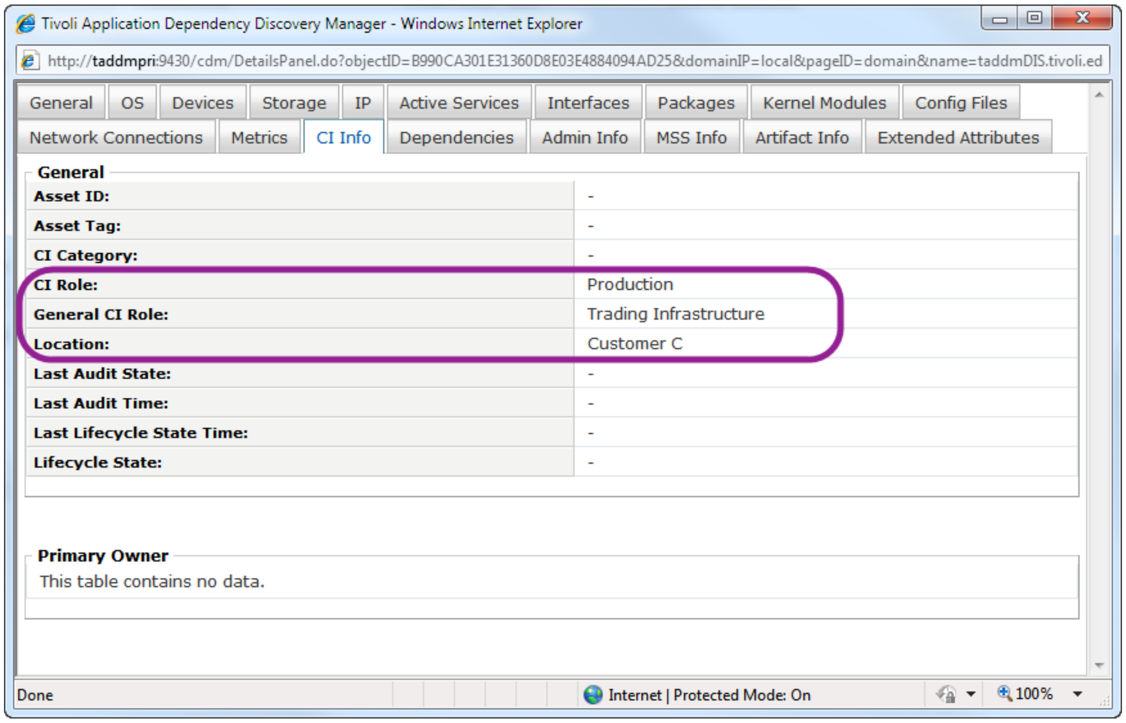
This line assigns the string *Location Name* as the label for the locationTag attribute.

e. Add any additional attribute labels you want to include.

f. Save the file.

3. To activate your modifications, restart the TADDM Storage Server.

After the server has restarted, you will see the locationTag information in the CI Info tab of the Details Pane. It will look similar to this example:


Tivoli Application Dependency Discovery Manager - Windows Internet Explorer
http://taddmpri:9430/cdm/DetailsPanel.do?objectID=B990CA301E31360D8E03E4884094AD25&domainIP=local&pageID=domain&name=taddmDIS.tivoli.ed
General
OS
Devices
Storage
IP
Active Services
Interfaces
Packages
Kernel Modules
Config Files
Network Connections
Metrics
CI Info
Dependencies
Admin Info
MSS Info
Artifact Info
Extended Attributes
Done
Internet | Protected Mode: On
100%

General

Asset ID:	-
Asset Tag:	-
CI Category:	-
CI Role:	Production
General CI Role:	Trading Infrastructure
Location:	Customer C
Last Audit State:	-
Last Audit Time:	-
Last Lifecycle State Time:	-
Lifecycle State:	-

Primary Owner

This table contains no data.

Notice that in addition to the locationTag, in this example additional template extensions have been used to populate the CIRole and generalCIRole attributes.

6 Summary

The location tagging feature of TADDM provides the means to:

- Isolate credentials between different organizational units (locations)
- Automatically populate the locationTag attribute of all discovered resources

Before enabling the location tagging feature consider the following:

- Locations are determined by the location associated with your anchor hosts. Anchor host locations must be manually maintained by modifying the `anchor.properties` file.
- During discovery, only credentials that have been associated with the location of the anchor host that performs discovery will be used to identify valid credentials and to populate the locationTag attributes.
- If the anchor host has no location, the default location specified in the com.ibm.cdb.locationTag property in the collation.properties file on the TADDM Discovery Server will be used to identify valid credentials and to populate the locationTag attributes.
- To avoid extensive warning messages, both com.ibm.cdb.location taggingEnabled and com.ibm.cdb.locationTag properties should be specified – or commented out.
- If location tagging is enabled, you cannot populate the locationTag attribute through custom template extensions.
- Out-of-the-box the locationTag attribute is not surfaced in the GUI, and cannot be used to create ad-hoc reports using the Query Wizard.
- Default TADDM reports do not use the locationTag attribute.
- The locationTag attribute is not defined in the TADDM Cognos namespace.

Appendix A.: Location tagging.py

The following jython script can be used to extend your Computer System or Custom Server Templates in order to populate the locationTag attribute without enabling the built-in location tagging feature.

To use it, copy the script to a file named `Location tagging.py` in the `$COLLATION_HOME/dist/etc/templates/commands/extension-scripts` directory, and add the following line to each of the template extensions for which you want to populate the locationTag attribute.

```
SCRIPT:etc/templates/commands/extension-scripts/Location tagging.py
```

Location tagging.py

```
############### Begin Standard Header - Do not add comments here ###############
#
# File:     %W%
# Version:  %I%
# Modified: %G% %U%
# Build:    %R% %L%
#
# Licensed Materials - Property of IBM
#
# Restricted Materials of IBM
#
# 5724-N55
#
# (C) COPYRIGHT IBM CORP. 2007.  All Rights Reserved.
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
############################# End Standard Header ##############################

################################################################################
#
#  This TADDM custom server extension has been designed to emulate the functionality
#  of the TADDM location tagging feature to populate the locationTag attribute of computerSystem
#  by assigning the location that is associated with the anchor host that support the scope in
#  in which the ipAddress of the computerSystem is included. If ipAddress belongs to multiple
#  scopes, and thereby can be discovered through multiple anchor hosts, the location of the
anchor
#  host with the highest suffix will be populated into the value of the locationTag attribute.
#
#  If the TADDM location tagging feature is enabled for the current Discovery Server, the value
to be assigned to
#  the locationTag attribute is controlled by TADDM, and therefore this script terminates
(almost)
#  immediately if it is discovered that location tagging is enabled.
#
#  If the TADDM location tagging is disabled, and the com.ibm.cdb.locationTag property has been
specified, you
#  will many warning messages in the log files. For that reason, this script uses  a new
property:
#
#                com.ibm.cdb.customLocationTag=<your default location>
#
#  to allow you to specify a default location that will be assigned to computerSystem that are
relate to the local anchor,
#  or is not included in a scope that can be associated with a specific anchor host. If the
com.ibm.cdb.customLocationTag
#  property has not been specified, the value 'UNKNOWN' will be used.
#
##################################################################################
import sys
import java
from java.lang import *
from java.io import *



### initialize environment when running under TADDM Control
try:

    collation_home = System.getProperty("com.collation.home")

    System.setProperty("jython.home", collation_home + "/external/jython-2.1")
    System.setProperty("python.home", collation_home + "/external/jython-2.1")

    jython_home = System.getProperty("jython.home")
    sys.path.append(jython_home + "/Lib")
    sys.path.append(collation_home + "/lib/sensor-tools")
    sys.prefix = jython_home + "/Lib"

    ## TADDM imports
    import sensorhelper

except:
    ##  initialization for the non-TADDM (testing)
    pass


## These jython classes can only be loaded after the jython environment has been initialized
import traceback
import string
import re
import StringIO
import jarray
import commands


###############################################################################
###############################################################################
###############################################################################
##
##   COMMON TADDM FUNCTIONS
##
###############################################################################
###############################################################################
###############################################################################


################################
## LogError      Error logger
################################
def LogError(msg):
    '''
    Print Error Message using Error Logger with traceback information
    '''
    try:
        log.error(msg)
    except:
        print msg

    (ErrorType, ErrorValue, ErrorTB) = sys.exc_info()
    traceback.print_exc(ErrorTB)
```

```
##############################################################
## LogDebug      Print routine for normalized messages in log
##############################################################
def LogDebug(msg):
    '''
    Print Debug Message using debug logger (from sensorhelper)
    '''
    # assuming SCRIPT_NAME and template name are defined globally...
    # point of this is to create a consistent logging format to grep
    # the trace out of
    msg = "\t" + msg
    try:
        log.debug(msg)
    except:
        if (logLevel == "DEBUG") :
            print msg
########################################################
# LogInfo Print routine for normalized messages in log
########################################################
def LogInfo(msg):
    '''
    Print INFO level Message using info logger (from sensorhelper)
    '''
    # assuming SCRIPT_NAME and template name are defined globally...
    # point of this is to create a consistent logging format to grep
    # the trace out of
    try:
        log.info(msg)
    except:
        print msg



###############################################################################
###############################################################################
###############################################################################
##
##   PRIVATE FUNCTIONS
##
###############################################################################
###############################################################################
###############################################################################


###########################################################
##  define True and False (not supported in Jython 2.1
###########################################################
def true_and_false():
    global False
    global True

    try:
        True and False
    except NameError:
        class bool(type(1)):
            def __init__(self, val=0):
                if val:
                    type(1).__init__(self, 1)
                else:
                    type(1).__init__(self, 0)
            def __repr__(self):
                if self:
                    return "True"
                else:
                    return "False"

            __str__ = __repr__
        bool = bool

        False = bool(0)
        True = bool(1)

############################################################
##  read property files
############################################################
def readPropFile(prop_file_name):

        #Get the CDM FileSystemContent object for TADDM's collation.properties
        ##coll_props_file = sensorhelper.getFile(collation_home + "/etc/collation.properties");
        LogDebug("Reading properties from: " + prop_file_name)


        f = open(prop_file_name, 'r')
        in_stream = str(f.read())
        f.flush()
        f.close()


        #Create a Java StringBufferInputStream attached to the captured collation.properties
        ##propStream = java.io.StringBufferInputStream(x)
        propStream = java.io.StringBufferInputStream(in_stream)

        #Create and load a Java Properties object
        props = java.util.Properties()
        props.load(propStream)


        ## Testing

        keys = props.keySet()
        LogDebug("Found " + str(keys.size()) + " properties in " + prop_file_name)
        for key in keys:
            LogDebug("read key: " + key + ": " + props.getProperty(key))


        return(props)

############################################################
##  run a command on the TADDM Discovery Server
############################################################
def runCommand(command):
    try:
        p = Runtime.getRuntime().exec(command);

        #BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.getInputStream()));
        #BufferedReader stdError = new BufferedReader(new InputStreamReader(p.getErrorStream()));



        stdInput = BufferedReader(InputStreamReader(p.getInputStream()));
        stdError = BufferedReader(InputStreamReader(p.getErrorStream()));


        LogDebug("StdOut from the command:\n");
        sysout = []
        s = ""
        while s != None :
            s = stdInput.readLine()
            if s == None:
                break
            LogDebug(s);
            sysout.append(s)

        LogDebug("StdErr from the command:\n");
        syserr = []
        e = ""
        while e != None:
            e = stdInput.readLine()
            if e == None:
                break
            LogDebug(e)

    except IOException, e:

        ##catch (IOException e) {
            LogError("exception happened - here's what I know: ");
            e.printStackTrace();

    return(sysout, syserr)


##############################
# get_ip_netmask        parses a scope element of the form
#                       ipaddress/netmask
#                       into a python sequence of (ip,netmask)
#                       also, if the netmask does not seem valid
#                       the netmask is set to 255.255.255.255
##############################
def get_ip_netmask(scope_element):

    #first see if the scope element consists of an address and a netmask
    if re.search(r'\/', scope_element) != None:
        #it does
        (ip, netmask) = string.split(scope_element, '/')
        if re.search(r'\d+\.\d+\.\d+\.\d+', netmask) == None:
            LogDebug(netmask, " is invalid; setting Netmask to 255.255.255.255 for ", ip)
            netmask = '255.255.255.255'
    else:
        #it does not
        ip = scope_element
        netmask = '255.255.255.255'
    return(ip, netmask)

##############################
# get_scope_type        determines the type of the provided scope
#                       either, address, subnet, or range
##############################
def get_scope_type(scope_element):

    #See if it has a '-' in it, if so then it is range
    if re.search(r'-', scope_element) != None:
        return('range')

    #Now it is either a subnet scope or a ip scope, call get_ip_netmask
    #to figure it it out
    (ip, netmask) = get_ip_netmask(scope_element)
    if netmask == '255.255.255.255':
        return('address');

    #must be a subnet
    return('subnet')


###############################################
#  call healthcheck to get the scopes
###############################################
def get_scopes_from_healthcheck():
    LogDebug("Executing command: " + healthCheck_command)

    output, syserr = runCommand(healthCheck_command)

    #scopes = {}
    taddm_scopes = []
    i = 0

    for line in output:
        if logLevel == "DEBUG":
            LogDebug ("read line " + str(i) + "  " + line.rstrip())

        inp = line.rstrip()

        #  filter out the header lines
        if (inp[:14] != "GROUP:  config" and inp != "" and inp[:2] != "**" and inp[:23] !=
"Label,Value,Description"):

            s = string.split(inp, ",")
            scope_name = string.strip(s[0])
            scope_elements = string.strip(s[1])

            if scope_elements != "":
                taddm_scope = scope(scope_name)
                taddm_scopes.append(taddm_scope)

                for element in string.split(scope_elements, "|"):
                    element = string.strip(element)
                    if element != "":
                        LogDebug("Adding element '" + str(element) + "' to scope ' " +
taddm_scope.getName())
                        taddm_scope.addElement(element)

    return taddm_scopes


###############################################
## read scopes from scope.properties
###############################################
def get_scopes_from_scope_properties(scope_prop_file_name):
    props = readPropFile(scope_prop_file_name)
    taddm_scopes = []
    for scope_name in props.keys():
        LogDebug("Processing scope: " + scope_name)
        scope_elements = props[scope_name]

        if scope_elements != "":
            taddm_scope = scope(scope_name)
            taddm_scopes.append(taddm_scope)

        #if string.find(scope_elements,",") > -1:
            for element in string.split(scope_elements, ","):
                element = string.strip(element)
                if element != "":
                    LogDebug("Adding element '" + str(element) + "' to scope ' " +
taddm_scope.getName())
                    taddm_scope.addElement(element)

    return taddm_scopes


################################################################################
##  class to support scopes
################################################################################
class scope:

    def __init__(self, name):
        self.__name__ = name
        self.__elements__ = {}


    def hasName(self):
        hasName = False
        if self.__name__ != None:
            hasName = True
        return hasName

    def getName(self):
        return self.__name__

    def setName(self, name):
        self.__name__ = name

    def addElement(self, element):
        method = "include"
        x = len(element) - 1
        if element[:1] == "(" and element[x:] == ")":
            method = "exclude"
            element = element[1:x]

        e = None
        if self.hasElement(element) == False:
            e = scopeElement(self.__name__,element, method)
            self.__elements__[element] = e
        elif method == "exclude":   # force exclude if the element has already been created
            e = self.getElement(element)
            if e.getMethod() != method:
                e.setMethod(method)

        return e

    def hasElement(self, element):
        return self.__elements__.has_key(element)

    def hasElements(self):
        hasElements = False
        if len(self.__elements__) > 0:
            hasElements = True
        return hasElements

    def getElements(self):
        return self.__elements__.values()

    def getElement(self, element):
        return self.__elements__[element]


###############################################################################
###############################################################################
###############################################################################
##
##   PRIVATE CLASSES
##
###############################################################################
###############################################################################
###############################################################################



###############################################################################
##  class to support scope-elements
###############################################################################
class scopeElement:

    def __init__(self, scope, name, method):
        self.__parent__ = scope
        self.__name__ = name
        self. __method__ = method
        self.__type__ = get_scope_type(name)
        self.__network__ = None
        self.__mask__ = None
        self.__rangeStart__ = None
        self.__rangeEnd__ = None

        if self.__type__ == "subnet":
            try:
                (self.__ip__, self.__mask__) = string.split(self.__name__, "/")
                self.__network__ = sensorhelper.calcNetworkAddress(self.__ip__, self.__mask__)
            except:
                pass
        elif self.__type__ == "range":
            (self.__rangeStart__, self.__rangeEnd__) = string.split(self.__name__, "-")

    def isIncluded(self, ip, mask=None):
        isIncluded = False
        if self.containsIp(ip) and self.__method__ == "include":
            isIncluded = True
        return isIncluded

    def isExcluded(self, ip):
        isExcluded = False
        if self.containsIp(ip) and self.__method__ == "exclude":
            isExcluded = True
        return isExcluded

    def containsIp(self, ip):
        containsIp = False


        if self.__type__ == "address":
            if ip == self.__name__:
                containsIp = True

        if self.__type__ == "subnet":
            network = None
            try:
                network = sensorhelper.ipInSubnet(ip, self.__network__, self.__mask__)
            except:
                network="192.168.1"

            if network == self.__network__:
                containsIp = True


        if self.__type__ == "range":
            (i1, i2, i3, i4) = string.split(ip, ".")
            (s1, s2, s3, s4) = string.split(self.__rangeStart__, ".")
            (e1, e2, e3, e4) = string.split(self.__rangeEnd__, ".")

            if i1 >= s1 and i1 < e1:
                containsIp = True
            elif  i1 == s1 and i1 == e1:
                if i2 >= s2 and i2 < e2:
                    containsIp = True
                elif  i2 == s2 and i2 == e2:
                    if i3 >= s3 and i3 < e3:
                        containsIp = True
                    elif  i3 == s3 and i3 == e3:
                        if i4 >= s4 and i4 <= e4:
                            containsIp = True

        return containsIp


    def hasName(self):
        hasName = False
        if self.__name__ != None:
            hasName = True
        return hasName
    def getName(self):
        return self.__name__

    def setName (self, name):
        self.__name__ = name

    def hasMethod(self):
        hasMethod = False
        if self.__method__ != None:
            hasMethod = True
        return hasMethod

    def getMethod(self):
        return self.__method__

    def setMethod(self, method):
        self.__method__ = method

    def hasType(self):
        hasType = False
        if self.__type__ != None:
            hasType = True
        return hasType

    def getType(self):
        return self.__type__

    def setType(self, elem_type):
        self.__type__ = elem_type




##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
#
#   MAIN
#
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################
##############################################################################


# include definition of True and False (not included in Jython 2.1
true_and_false()


LogInfo(" ====== STARTING location Tagging script ====== ")

try:
    # The first thing we need to do is get the Objects that are passed to a sensor
    (os_handle, result, server, seed, log) = sensorhelper.init(targets)
    # os_handle  -->  Os handle object to target system
    # result     -->  Results Object
    # server     -->  AppServer or ComputerSystem Object that is discovered
    # seed       -->  seed object which contains information that was found prior
    #                 to this CSX running
    # Logger     -->  Object to write to sensor log
    addr = targets.get("IpAddressSeed")          # seed object which contains information that
was found prior
    LogInfo("seed: " + str(seed) + "     " + seed.getClass().getName())               # to this
CSX running
    LogInfo("addr: " + str(addr))               # to this CSX running

    IpAddress = str(seed)
    LogInfo("IpAddress: " + IpAddress)

    LogInfo(" ====== On IP Address: " + str(seed) + " ====== ")
    LogInfo(" ====== Using sensorhelper version: " + str(sensorhelper.getVersion()))
    LogDebug(" ====== Jython sys.path=" + str(sys.path))
    LogDebug(" ====== Jython script_dep.jy: sys.prefix=" + str(sys.prefix))


    api_major_version = sensorhelper.getApiMajorVersion()
    api_minor_version = sensorhelper.getApiMinorVersion()
    LogDebug("sensorhelper version is " + str(api_major_version) + "." + str(api_minor_version))
    LogDebug("TADDM is version " + sensorhelper.getTADDMVersion())

    healthCheck_command = collation_home + "/bin/healthcheck  -c checkTaddmScopes"


except:

    ## The following variables are used to simulate expected TADDM behavior
    ## so the script can be tested without connecting to TADDM
    logLevel = "INFO"
    collation_home = "C:/dev/taddm7213_sdk/sdk/etc"
    IpAddress = "192.168.81.135"
    healthCheck_command = "cat \"" + collation_home + "/etc/healthcheck_checkTaddmScopes\""



try:
    #######################################
    ## Get the collation_home property
    ######################################
    LogDebug("Collation home is: " + str(collation_home))
    if collation_home == None:
        LogError("Unable to locate com.collation.home");
        raise RuntimeError, "Unable to locate com.collation.home"

    # get the default location
    coll_props_file_name = collation_home + "/etc/collation.properties";

    props = readPropFile(coll_props_file_name)

    #################################################################
    ##   Leave if location tagging is enabled
    #################################################################

    location taggingEnabled = props.getProperty("com.ibm.cdb.location taggingEnabled")
    if location taggingEnabled != None:
        if string.upper(location taggingEnabled) == "TRUE" :
            LogInfo("*** Location tagging is enabled - leaving")
            sys.exit()


    #################################################################
    ##  set default location
    #################################################################
    locationTag_property = "com.ibm.cdb.customLocationTag"
    defaultLocation = props.getProperty(locationTag_property)
    if (defaultLocation != None):
        LogDebug("Default location from " + locationTag_property + " property is: " +
defaultLocation)
```

```
    else:
        LogDebug("Unable to locate " + locationTag_property + " property - assuming defaultValue
of UNKNOWN")
        defaultLocation = "UNKNOWN"

    ##########################################################################################
    ##  Get the scopes that are defined to the discovery server:
    ##
    ##  Scopes can be read from either the scopes.properties file, of from the output of
    ##  the 'healthcheck checkTaddmScopes' command
    ##
    ##  The healthcheck command method is included to demonstrate how to execute a command
    ##  on the Discovery Server and capture the output
    ##########################################################################################
    #taddm_scopes = get_scopes_from_healthcheck()
    taddm_scopes = get_scopes_from_scope_properties(collation_home + "/etc/scope.properties")

    #########################################################
    ## find the scopes that the current IpAddress belongs to
    ## by reading the scopes.properties file, and use the
    ## private scope and scopeElement classes to find the
    ## scope
    #########################################################
    included_in_scopeElements={}
    excluded_in_scopeElements={}

    # loop over scopes, extract the scope-elements, and validate if the IpAddress is included
    scope_elements = []
    included_in_scopes = {}
    for scope in taddm_scopes:
        scope_name = scope.getName()
        excluded = False
        for scope_element in scope.getElements():
            included = False
            scope_element_name = scope_element.getName()
            scope_element_type = scope_element.getType()
            scope_and_element_name = scope_name + ":" + scope_element_name
            LogDebug("Analyzing element: " + scope_and_element_name)

            if scope_element.isExcluded(IpAddress):
                excluded = True
                LogInfo("*** " + IpAddress + " is EXCLUDED from scope element " + scope_name +
":" + scope_element_name)
                #excluded_in_scopeElements[scope_and_element_name]=True
            elif scope_element.isIncluded(IpAddress):
                included = True
                LogInfo("*** " + IpAddress + " is INCLUDED in scope element " + scope_name + ":"
+ scope_element_name)
                #included_in_scopeElements[scope_and_element_name]=True
                included_in_scopeElements[scope_and_element_name] = scope_name

            #  bypass further processing if this scope if the IpAddress exists in an EXCLUDE
scope element
            if excluded:
                break

        # build a dictionary of scopes that include the IpAddress
        for element_name, scope_name in included_in_scopeElements.items():
            if not included_in_scopes.has_key(scope_name):
                included_in_scopes[scope_name] = element_name

    # Print a nice message that shows which scopes the IpAddress belongs to
    incl = None
    for scope_name in included_in_scopes.keys():
        if incl == None:
            incl = "'" +scope_name+"'"
        else:
            incl = incl + ", '" + scope_name+"'"
    LogInfo("Computer System with IpAddress " + IpAddress + " is incluced in " +
str(len(included_in_scopes)) + " socpes: " + incl)

    ##############################################################
    ## find the anchor hosts from the anchor.properties file
    ## and identify related scopes and locations
    ##############################################################
    anchor_props_file_name = collation_home + "/etc/anchor.properties";
    anchor_props = readPropFile(anchor_props_file_name)

    ## find the anchor hosts, scopes and locations
    ## and save them in three dicts using the anchor number as key
    anchor_hosts = {}
    anchor_scopes = {}
    anchor_locations = {}
    anchor_locations["0"] = defaultLocation
    anchor_host_prefix = "anchor_host_"
    hst = len(anchor_host_prefix)
    anchor_scope_prefix = "anchor_scope_"
    sco = len(anchor_scope_prefix)
    anchor_location_prefix = "anchor_location_"
    loc = len(anchor_location_prefix)

    for prop in anchor_props.keys():
        if prop[:hst] == anchor_host_prefix:
            anchor = prop[hst:]
            anchor_hosts[anchor] = anchor_props[anchor_host_prefix+anchor]
            val = anchor_props[anchor_scope_prefix+anchor]
            if val == None:
                val = ""
            anchor_scopes[anchor] = val
            if anchor != "0":
                val = anchor_props[anchor_location_prefix+anchor]
                if val == None:
                    val = ""
                anchor_locations[anchor] = val


    # sort the anchors
    anchors = anchor_hosts.keys()
    anchors.sort()

    # print anchor scope assignments
    for anchor in anchors:
        LogInfo("*** Anchor 'anchor_host_" + str(anchor) + "' is assigned scope '" +
anchor_scopes[anchor] + "' and location '"  + anchor_locations[anchor]+ "'")


    ###############################################################
    ##  assign location based on scope-anchor-location assignment
    ##
    ##  use the ascending sequence so that the highest anchor number
    ##  that supports a scope is assigned
    ###############################################################
    location = defaultLocation

    i = len(anchors) - 1

    while (i >= 0):
        anchor = anchors[i]
        scope = anchor_scopes[anchor]

        if included_in_scopes.has_key(scope):
            location = anchor_locations[anchor]
            break
        i = i-1;
    LogInfo("ComputerSystem with IpAddress " + IpAddress + " will be assigned a location of: " +
str(location))
```

```
finally:
    pass

##########################################################################
##########################################################################
##   assign a value ot the locationTag attribute,
##   and save the results
##########################################################################
##########################################################################
try:
    server.setLocationTag(location)
    result.setComputerSystem(server)

except:
    pass
```

Notices

Trademarks