CIMERA INTEGRATIONS

Integrating with Subversion

A case study

Version 1.0, 11-Jan 2016

Jon Bentley, Gwyn Carwardine

jon.bentley@propelsystems.com
gwyn.carwardine@propelsystems.com

Contents

1 Requirement

The requirement is to integrate the open-source SCM (Software Configuration Management) tool "Subversion" with Cimera such that builds created in Subversion are reconciled with builds recorded in Cimera. Additional information held against the builds recorded in Subversion is to be copied into Cimera (the Subversion revision number and the date/time the build was created).

It is required to view builds created in subversion from within Cimera, the "single source of truth". Builds recorded in Cimera can then be related to other items in Cimera, providing additional management and impact assessment capabilities.

The integration is to work automatically (requiring no user activity) and new builds created in Subversion should be available in Cimera within a short period, measured in minutes rather than hours.

There is one Cimera service but there may be multiple Subversion repositories hosted on different servers.

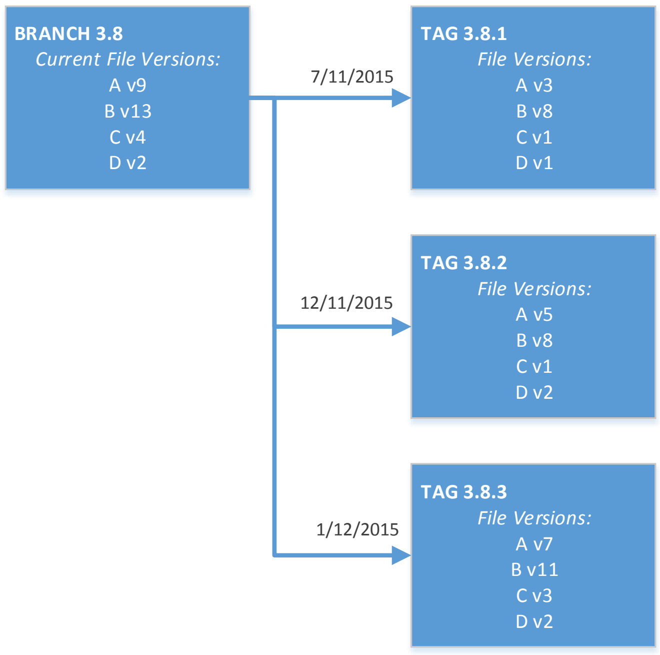
Any problems during the automated reconciliation process should be recorded and notified to the appropriate person.

2 Background

2.1 Subversion

The open-source SCM (Software Configuration Management) product Apache Subversion, with companion product TortoiseSVN, is used within the organisation to manage all source code. Source code is managed using "branches" for development and unit testing is performed within these branches. When the code is considered fit for release to formal testing (system / integration / acceptance test etc) a "tag" is created. A tag is effectively a snapshot, or list, of the current version of each source code file at that point in time and represents a "build". It is these builds that are then promoted through the testing lifecycle and, if signed off, released into production.

The following diagram shows Branch 3.8, its current versions, and the three Tags that have been "snapshotted" at various times.



Note: Subversion is a very flexible tool and can be configured and used in many different ways to suit a particular organisation's development process. There is no "one size fits all" model.

Subversion runs on both Windows and UNIX platforms.

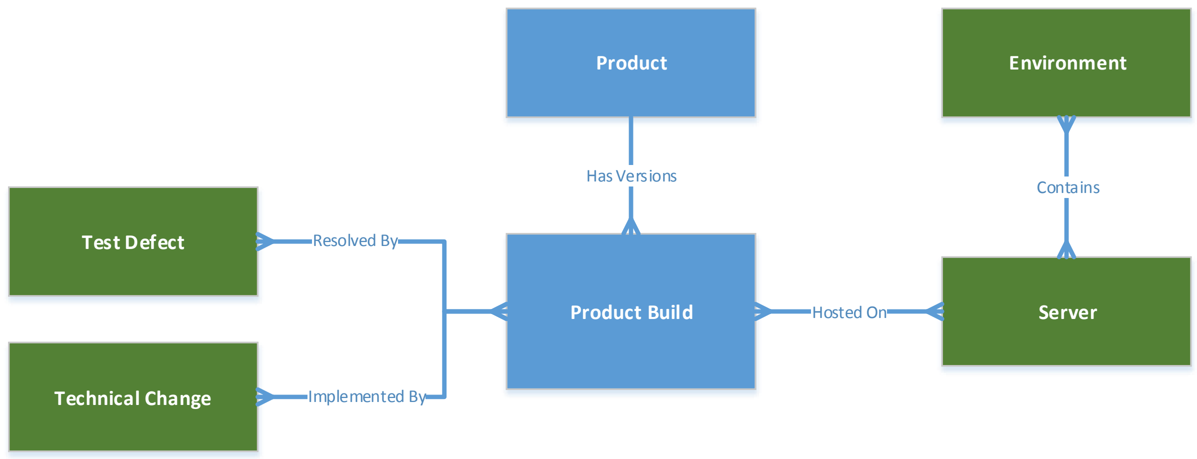
2.2 Cimera

Cimera is used to track all Products and individual Product Builds (versions) within the organisation. Cimera is also used to manage many other types of items including Test Defects and Changes that it is desirable to link to Product Builds to answer questions such as "what changes are included in this Product Build?" and "what defects does this Product Build resolve?"

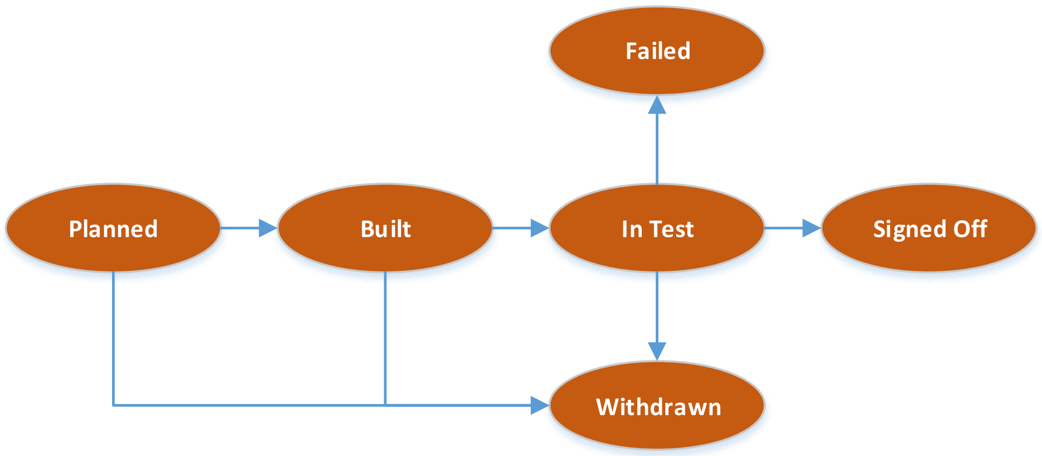
Cimera additionally records deployments of Product Builds to Environments and Servers answering the questions "where is this Product Build installed?" and "what Product Builds are installed on this server?"

Using Cimera's audit facilities it is also possible to determine answers to questions such as "what Product Build was present in this Environment on this date?" and "what other product version were present at that time?"

A relevant subset of the organisation's specific Cimera configuration is shown below:



A Product Build has been configured with the following lifecycle:



A Product Build begins life in the "Planned" state. At this point the Build is registered, even if development work has not yet started. In this planning state defects and changes can be related to the Build – providing a recipe for what defects this Build is intended to resolve and what technical changes it is intended to deliver.

When development has completed and the code has been unit tested it is then built, ready for deployment and testing, and the Build is promoted to the "Built" status within the lifecycle.

Note: Like Subversion, Cimera is a very flexible tool and can be configured and used in many different ways to suit a particular organisation's development process. There is no "one size fits all" model.

Cimera runs on the Windows platform.

3 Integrating Cimera and Subversion

Both Cimera and Subversion are unpredictable in terms of their use cases and configuration. There is no standard "connector" available as it would either only work in very limited cases or have to be very highly configurable and even then would not be guaranteed to be relevant in all situations.

It is therefore easier, and more likely to meet the organisation's need to create a custom integration, interfacing to both systems.

3.1 Interfacing with Subversion

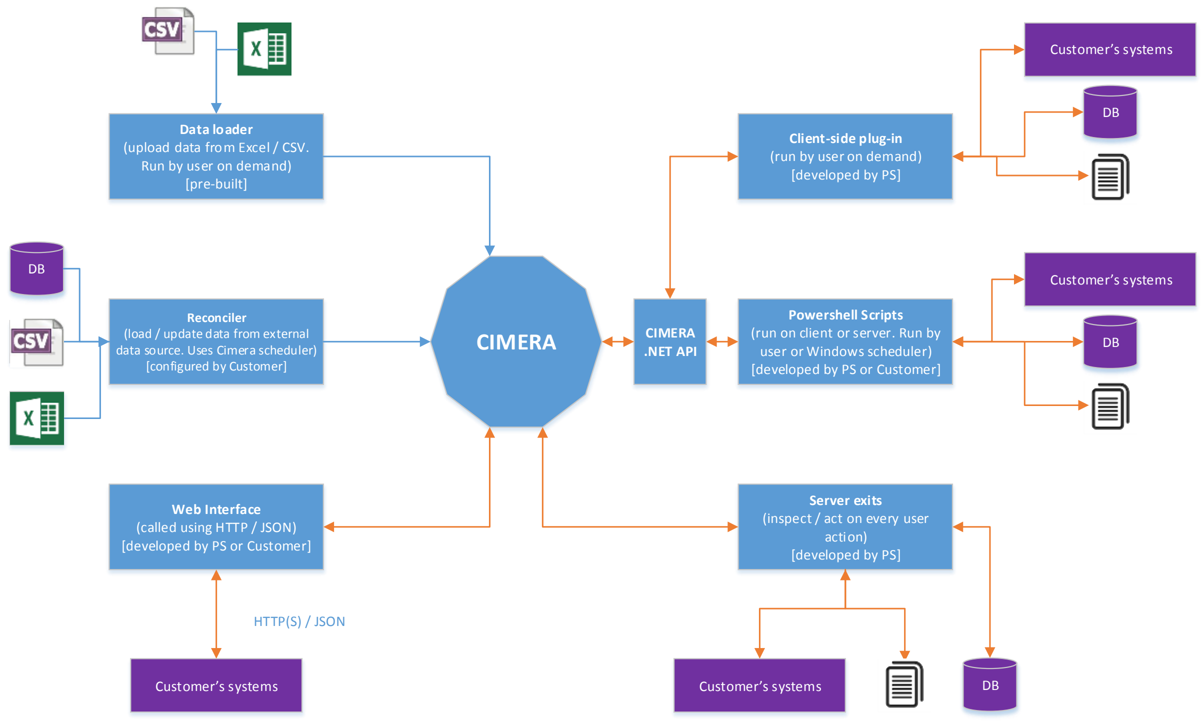
Subversion has a command line client that can be used to perform all client functions, including extracting metadata from the server's repository.

Subversion also supports "Repository Hooks". These are server side scripts that are called when something changes in the Subversion repository. They can inspect the changes and either reject the changes or be used to provide notification to users or external systems.

There is no approved method to access Subversion's underlying database.

3.2 Interfacing with Cimera

Cimera has many ways in which it can be integrated with external tools, systems and data sources:



CSV
Data loader (upload data from Excel / CSV. Run by user on demand) [pre-built]
DB
CSV
Reconciler (load / update data from external data source. Uses Cimera scheduler) [configured by Customer]
CIMERA
CIMERA .NET API
Client-side plug-in (run by user on demand) [developed by PS]
Customer's systems
DB
Powershell Scripts (run on client or server. Run by user or Windows scheduler) [developed by PS or Customer]
Customer's systems
DB
Web Interface (called using HTTP / JSON) [developed by PS or Customer]
HTTP(S) / JSON
Customer's systems
Server exits (inspect / act on every user action) [developed by PS]
Customer's systems
DB

It is a matter of selecting the most appropriate interface in conjunction with the other system's available interface.

Criteria taken into consideration should be:

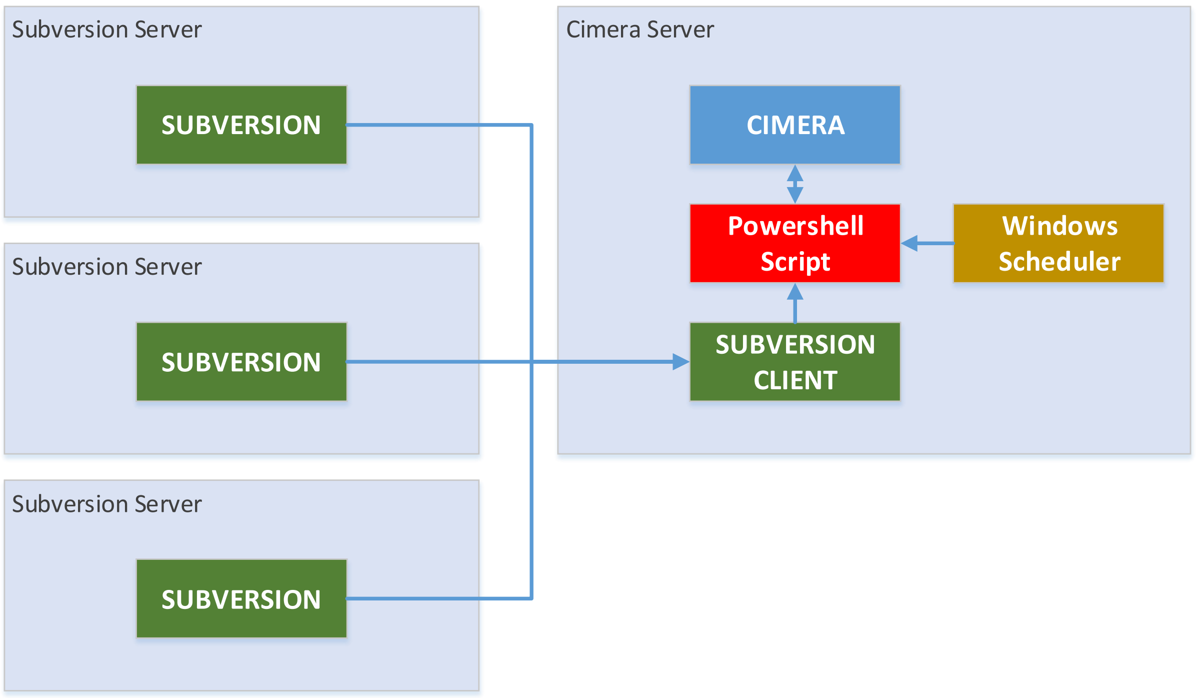
- Simplicity – Simplicity of the integration and its deployment and configuration will make the integration easier for people to understand and maintain in the future.

- Robustness – Any integration should be reliable and any errors should be recorded and notified to the responsible person or team. Fault tolerance is important and minimal manual activity should be required after failure.
- Skills – Does the organisation's skills better suit using one approach or another? In the future will the organisation have access to the skills necessary for maintenance?
- Approved API / Access – It is desirable to integrate using published and supported methods and APIs (Application Programming Interfaces). Approved APIs will normally continue to be supported in new versions of the product. Sometimes this is not possible with legacy products or those that do not provide APIs.
- Cross-Platform – If the products to be integrated sit on different platforms then it may eliminate some of the integration options as they cannot be used across different platforms.
- User-dependency – Does the solution have a dependency on user action? Sometimes this is acceptable, or even a requirement (the integration must be invoked by a user – normally when some intelligence or approval is required), and sometimes it is desirable for an integration to operate quietly in the background.
- Consistency – if it is required to integrate across a number of systems or data sources then it may be desirable to use one consistent method rather than a number of different methods.

3.3 Integration Options

The following options were considered. This is not an exhaustive list but some options were discounted due to obvious complexity.

3.3.1 Option 1: Using Subversion command line interface



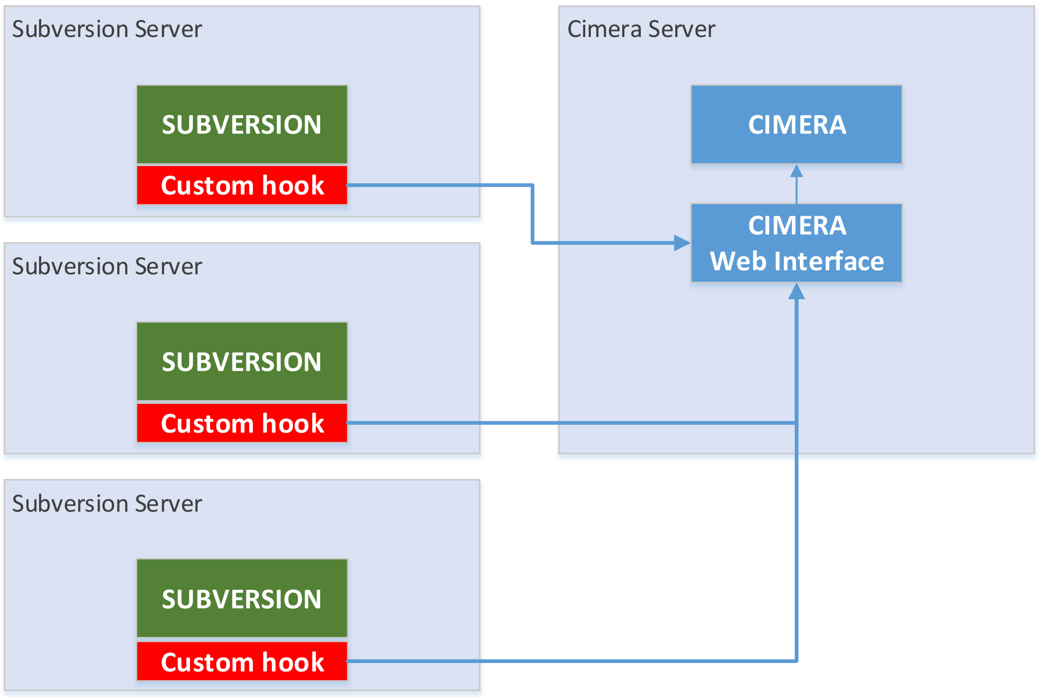
3.3.1.1 Description

A Powershell script is created that runs on the Cimera server. It calls the Subversion CLI (command line interface) which is also installed on the Cimera server to retrieve information from the different Subversion repositories and then reconciles this with information in Cimera using the standard

Cimera API. The Windows Scheduler is used on the Cimera server to manage running the Powershell script according to an appropriate schedule.

Advantages	Disadvantages
• Requires single deployment on the Cimera server. • Cimera API is easiest to code against robustly. • Subversion has a well documented and simple Command Line Interface requiring no specialist Subversion skills. • Powershell extracts all builds from Subversion and compares with Cimera so if there are any failures, the next time it runs it will catch up.	• Builds are not immediately reconciled with Cimera – if the script runs every 10 minutes then there could be up to 10 minutes before the information is reflected in Cimera.

3.3.2 Option 2: Using Subversion hooks and Cimera Web Interface



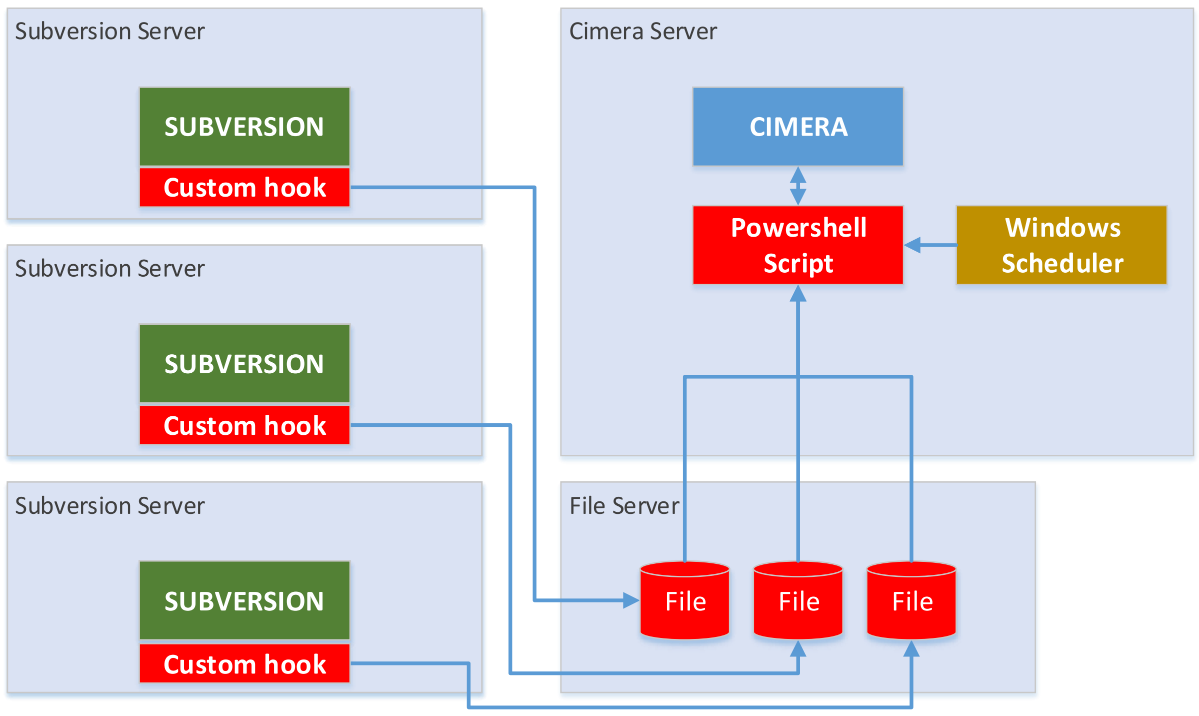
3.3.2.1 Description

A Subversion "Custom hook" is programmed that monitors Subversion activity and on creation or deletion of a "tag" calls Cimera via its Web Interface (as the Cimera API can only be called from machines running Windows and from programming languages that interface with .NET).

Advantages	Disadvantages
• Subversion activities are immediately reflected in Cimera.	• Requires more detailed Subversion knowledge and programming. • Hooks may have to be written differently on Windows and UNIX. • Multiple deployments.

	• Programming against the Cimera Web Interface is harder than the Cimera API. • Only changes are sent to Cimera so if the process fails then changes may be missed. This may require extra coding to ensure changes are not lost.

3.3.3 Option 3: Using Cimera hooks and intermediate files



3.3.3.1 Description

A subversion "Custom hook" is programmed to write Subversion activities to a file on a file server. The Windows Scheduler periodically runs a script to reconcile the contents of the files with Cimera.

Advantages	Disadvantages
• Uses simpler Cimera API. • Separation into Subversion and Cimera interface code – neither requiring knowledge of the other, only the shared file format.	• Builds are not immediately reconciled with Cimera – if the script runs every 10 minutes then there could be up to 10 minutes before the information is reflected in Cimera. • Hooks may have to be written differently on Windows and UNIX. • Multiple deployments. • Potential problems with file locking (if script is processing a file that is in use by Subversion) and file management. • Only changes are sent to Cimera so if the process fails then changes may be

	missed. This may require extra coding to ensure changes are not lost.

3.4 Selected Option

Option 1 was selected:

- The developer had good knowledge of Cimera but only basic knowledge of Subversion.
- There is a small library of Powershell Cimera helper functions and skeleton scripts.
- The slight delay in information being loaded into Cimera is acceptable and the schedule could be set to run every 5 minutes to reduce the puts little load on Subversion and Cimera.
- Single simple deployment on one server
- File based interfaces are simple and have been the de facto standard for many years but require extra effort to make resilient.

3.5 Solution Design

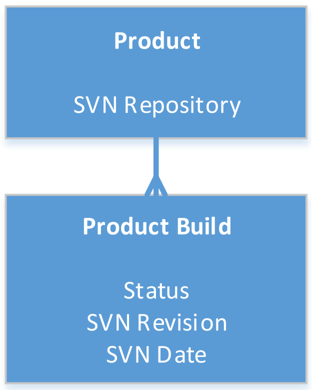
The solution requires changes to Cimera to add new attributes to store the Subversion information.

3.5.1 Cimera Configuration

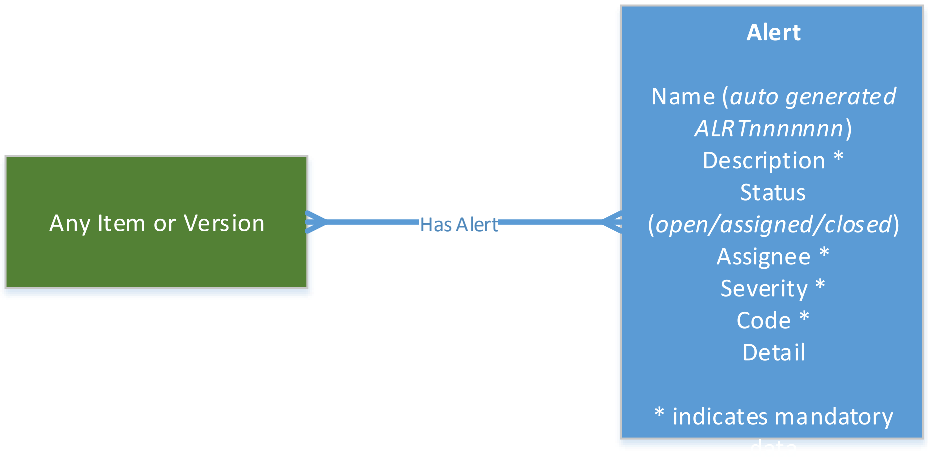
Add a text attribute "SVN Repository" to the Product (stem). This is used to indicate if the Product is managed in Subversion and if so where the repository is (repository name is a URL, e.g. https://mysvnserver.com/svn/product)

Add an integer attribute "SVN Revision" to the Product Build (version). This is used to store the Subversion revision number that relates to the build. The script will populate this and will ensure it is consistent. If for any reason it is changed in Subversion the change will be reflected in Cimera.

Add a Date Time attribute "SVN Date" to the Product Build (version). This is used to store the Date/Time when the Subversion build was created. The script will populate this and will ensure it is consistent. If for any reason it is changed in Subversion the change will be reflected in Cimera.



When an error is detected in the script it will also create an item of type "Alert" in Cimera and assign it to a resolving group. Therefore need to add a new Item Type of Alert and supporting fields. This is only required because Alerts have never been required or defined in Cimera before. This is an extension that may well find other uses where automated processing or other interfaces need to notify an appropriate person or team that a failure has occurred. Using alerts also means that if the script runs once and finds a problem, the next time it runs it can check to see if there is an open alert for the problem and therefore does not need to raise it again.


Any Item or Version
Has Alert
Alert
Name (*auto generated ALRTnnnnnnn*)
Description *
Status (*open/assigned/closed*)
Assignee *
Severity *
Code *
Detail
* indicates mandatory data

3.5.2 Psuedo-code

The following psuedo-code describes the technical process:

```
Get a list of all Products (Product stem) from Cimera
For each Product
    If Product has SVN Repository defined then
        Call SVN to get a list of “tags”
        If Repository does not exist create an alert in Cimera
        Get a list of all Product Builds (Product versions) from Cimera
        For each SVN tag
            If a Product Build does not exist in Cimera then create an alert
            If the Cimera status is “Planned” then promote to “Built”
            If the SVN information is missing or incorrect in Cimera then set it
        Next
    End If
Next
```

3.6 Implementation

3.6.1 Full script

The full Powershell script (excluding Cimera function library):

```
$CimeraServer = "thruster.propelsystems.com"
$CimeraUser = 'xxx'
$CimeraPass = 'xxx'
$CimeraDB = 'UDMS'
$svnExe = "C:\Program Files\TortoiseSVN\bin\svn.exe"
$tagsFile = 'd:\cimera\scripts\tags.xml'
$svnLogFile = 'd:\cimera\scripts\svnLog.txt'
$svnUser = 'xxx'
$svnPass = 'xxx'

function DoWork()
{
  $cql = "find [product]"
  $products = Cimera-Query($cql)
  foreach ($id in $products.keys)
  {
    $product = $product.getciobject($id)
    if ($product.fields.exists("FLD:A1015") -and $product.fields.item("FLD:A1015").value -ne "")
      {
      $svnRepo = $product.fields.item("FLD:A1015").value
      If (Test-Path $tagsFile){Remove-Item $tagsFile}
      If (Test-Path $svnLogFile){Remove-Item $svnLogFile}
      $gotSvnDetails = $True
```

```
   try {
    & $svnExe ls --xml "$svnRepo" --username "$svnUser" --password "$svnPass"  > "$tagsFile" 2> "$svnLogfile"
   } catch {
    $gotSvnDetails = $False
    RaiseInvalidRepoAlert $id $svnRepo $_.exception.Message
   }
   if ($gotSvnDetails -eq $True)
   {
    $tags = [xml] (Get-Content $tagsFile)
    $cql = "find [product] version where civciid='" + $product.fields.item('FLD:A$CIOBJID').value + "'"
    $productVersions = Cimera-Query($cql)
    foreach ($tag in $tags.lists.list.entry)
    {
     "CHECKING " + $tag.name
     $matchingVersion = ""
     foreach ($vid in $productVersions.keys)
     {
      $productVersion = $productVersions.getciversion($vid)
      if ($tag.name -eq $productVersion.civersionname)
      {
       $matchingVersion = $productVersion
       break
      }
     }
     if ($matchingversion -eq "")
     {
      $tag.name + " not found"
      RaiseMissingBuildAlert $id $tag.name
     } else {
      $currentStatus = $matchingVersion.fields.item('FLD:O$CISTATUS').value
      $revDate = Cimera-ConvertToCimeraDateTime $tag.commit.date
      $revNo = $tag.commit.getattribute("revision")
      if ($currentStatus -eq 'STA:@16' `
        -or -not $matchingVersion.fields.exists('FLD:I1016') `
        -or -not $matchingVersion.fields.exists('FLD:A1017') `
        -or $matchingVersion.fields.item('FLD:I1016').value -ne $revNo `
        -or $matchingVersion.fields.item('FLD:A1017').value -ne $revDate)
      {
       $ExistingBuild = Cimera-GetItemForUpdate $vid
       if ($currentStatus -eq 'STA:@16')
       {
        $ExistingBuild.fields.item('FLD:O$CISTATUS').value = 'STA:@21'
       }
       $ExistingBuild.fields.item('FLD:I1016').value = $revNo
       $ExistingBuild.fields.item('FLD:A1017').value = $revDate
       $ExistingBuild = Cimera-UpdateItem $ExistingBuild
      }
     }
    }
   }
  }
 }
}

function RaiseMissingBuildAlert($rProductId,$rMissingBuildName)
{
 #Is there already an alert?
 $cql = "find alert where Code = 1 and cidescription = '$rMissingBuildName' and cistatus <> 'closed' and [Alert For](FIND
[Product] WHERE ciobjid= '$rProductId')"
 $existingAlerts = Cimera-Query($cql)
 if ($existingAlerts.count -eq 0)
 {
  # No, so create one
  $newAlert = Cimera-GetNewItem 'TYP:1011' 'BKT:@HELP'
  $newAlert.fields.item('FLD:A1012').value = "Error"
  $newAlert.fields.item('FLD:I1013').value = 1
  $newAlert.fields.item('FLD:O$CIOWNER').value = 'GRP:$ADMIN/'
  $newAlert.fields.item('FLD:O$CIASSIGNEE').value = 'GRP:$ADMIN/'
  $newAlert.fields.item('FLD:A$CIDESCRIPTION').value = $rMissingBuildName
  $newLink = Cimera-GetNewLink 'VLK:1013:R' $rProductId
  $newAlert.linksets.addlink($newLink, $False)
  $newAlert = Cimera-CreateItem($newAlert)
 }
}

function RaiseInvalidRepoAlert($rProductId,$rInvalidRepoName,$rErrorDetail)
{
 #Is there already an alert?
 $cql = "find alert where Code = 2 and cidescription = '$rInvalidRepoName' and cistatus <> 'closed' and [Alert For](FIND
[Product] WHERE ciobjid= '$rProductId')"
 $existingAlerts = Cimera-Query($cql)
 if ($existingAlerts.count -eq 0)
 {
  # No, so create one
  $newAlert = Cimera-GetNewItem 'TYP:1011' 'BKT:@HELP'
  $newAlert.fields.item('FLD:A1012').value = "Error"
  $newAlert.fields.item('FLD:I1013').value = 2
  $newAlert.fields.item('FLD:O$CIOWNER').value = 'GRP:$ADMIN/'
  $newAlert.fields.item('FLD:O$CIASSIGNEE').value = 'GRP:$ADMIN/'
  $newAlert.fields.item('FLD:A$CIDESCRIPTION').value = $rInvalidRepoName
  $newAlert.fields.item('FLD:T1014').value = $rErrorDetail
  $newLink = Cimera-GetNewLink 'VLK:1013:R' $rProductId
  $newAlert.linksets.addlink($newLink, $False)
  $newAlert = Cimera-CreateItem($newAlert)
 }
}
```

3.6.2 Skills and Effort required

Creating and implementing the solution required knowledge of the Cimera API, Cimera configuration and Powershell scripting. Little Subversion knowledge was required.

In terms of effort, the solution took the following time to develop and implement in production:

Solution Design	0.5 days
Develop and Test	0.5 days
Implement	0.5 days
TOTAL	**1.5 days**