



User Manual Administration
DELMIA Process Engineer®

Manufacturing accountability check



Contents

Manufacturing accountability check	1
Contents	2
Manufacturing accountability check	3
General	3
Server support for manufacturing accountability check	3
Action filter mode	4
User Exit framework	4
Type: actionproxy	4
Type: ergoproject	4
Customization	5
Limitation	5
Exemplary Manufacturing Accountability Check.....	5
Creating the UserExit DLL	7
Create Project	7
Build Project	7
Create User Exit Key in the Registry	8
Class XCMCCheckAcceptanceTest	8

Manufacturing accountability check

General

The manufacturing accountability check will be called each time, when the state of an action is updated. Only in the case when the check is successful, the state will be modified.

The manufacturing accountability check will be implemented by a customer within a DLL. In order to establish the call, a COM object server (out-of-process server) will be placed between IPD server and the customer DLL. The IPD server creates an instance of the COM server. The COM server loads the customer DLL and finally calls the execute function.

In order to perform the manufacturing accountability check a new filter mode is required that displays components used by a selected action irrespective of set planning state filter criteria.

This function will allow to

- Store information in a blob that is needed as input for the manufacturing accountability check
- Provide a framework in a form of a COM object server to implement a DLL (so-called User Exit).
- Call the execute function of a DLL in order to perform the check
- Display components used by a selected action irrespective of set planning state filter criteria.

Server support for manufacturing accountability check

In order to perform the manufacturing accountability check, input data is necessary. The input data will be stored into a blob and populated by web-services when the action state is updated.

The IPD server will provide accessor-methods to manage the content of this blob as a string. Its content and internal format will be specified by the customer who is implementing the User Exit.

When the update method for the action state is called, the IPD server checks in the Registry whether a User Exit for the manufacturing accountability check is defined. Only when a User Exit is defined the execute function will be called. Depending on the return value of the execute function, the transaction will be committed or not.

When the User Exit reports warnings, there will be a flag, that indicates whether the update of the action state will be performed or not. In any case, the warnings will be reported to the user.

Action filter mode

In order to perform the manufacturing accountability check a new filter mode is required that displays components used by a selected action irrespective of set planning state filter criteria.

This filter mode is triggered by the transient filter attribute **filter_action**.

When the attribute **filter_action** is set to the value true, the versions of components that belong to the current selected action pass the filter irrespective of set planning state filter criteria.

This filter mode is intended to be used only by manufacturing accountability check.

User Exit framework

This highlight provides a possibility to check the data consistency by the customer itself within a DLL (so-called User Exit). Instead of calling the Execute function directly by the IPD server, a COM object server (out-of-process server) will be placed between IPD server and customer DLL. The IPD server creates an instance of the COM server. The COM server loads the customer DLL and calls its Execute function. This indirect call reduces the possible impact on the IPD server from erroneous behaviour of the customer DLL.

The customer DLL will be restricted to read-only access to IPD data. An attempt to execute writing methods will be rejected with an error code.

The following data will be added to the data model

Type: actionproxy

The type will be extended by the following attribute

Name	Type	Description
externaldata	String ¹	External data input for manufacturing accountability check
userexiterrormsg	String	Error message reported by user exit
forceupdate	Bool	Flag to indicate that the update is enforced in case that user exit reports warnings
changeorderid	String	Top level change order identifier

Type: ergoproject

The type will be extended by the following attribute

Name	Type	Description
filter_action	Bool	Set action filter mode

¹ Stored as blob

Customization

Name and path of the customer DLL have to be specified by an entry in the windows registry.

In `HKEY_LOCAL_MACHINE\SOFTWARE\DELMIA\IPDSERVER\UserExit` the string “**ManufacturingAccountabilityCheck**” has to be created, which holds as value path and name of the DLL.

Limitation

The customer DLL will be restricted to read-only access to IPD data.

Exemplary Manufacturing Accountability Check

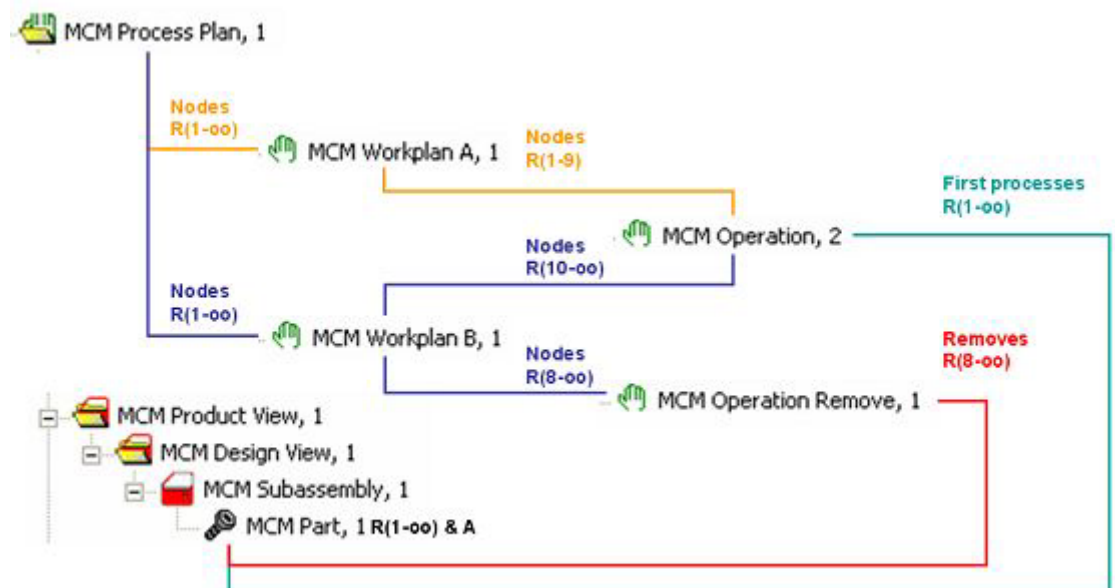
The purpose of the exemplary manufacturing accountability check is to show how the API of the IPD server can be used by the User Exit to implement one possible manufacturing accountability check. The manufacturing accountability check presented here doesn't claim to be complete and sufficient for all purposes.

The exemplary manufacturing accountability check will be performed in two steps.

1. Manufacturing accountability check regarding the manufacturing extended effectivity:
 - ➊ The check iterates thorough all parts listed in the external data field. For each part the extended effectivity is obtained
 - ➋ For a part all relations Process First Processes Product are collected. For each collected relations the extended effectivity is obtained
 - ➌ For a collected relation Process First Processes Product the process is retrieved
 - ➍ For each process, all relations Nodes are collected. For each collected relation Nodes the extended effectivity is obtained
 - ➎ For a collected relation Nodes the parent process is retrieved
 - ➏ The Steps d and e are repeated until a top level component of the process structure is reached
 - ➐ A logical expression is built according to the following rules:
 - For each single path from a part to a top level component obtained extended effectivities are connected by END to build a logical expression
 - The resulting expression is computed by connecting different paths by OR
 - ➑ The manufacturing accountability check is satisfied when the extended effectivity of a part is fully contained in the computed expression
2. Manufacturing accountability check regarding the engineering extended effectivity:

- The check iterates thorough all parts listed in the external data field. Furthermore, for each part the engineering extended effectivity is retrieved from the external data filed.
- The same procedure is repeated as for Step 1, but the relation Process Removes Product is used instead of Process First Processes Product. The expression is built.
- The resulting expression is computed by cutting the resulting expression of Step 1 by the resulting expression of Step 2
- The engineering accountability check is satisfied when the engineering extended effectivity of a part is logically equal to the computed expression

The following figure shows an example of a process and product structure for which the manufacturing accountability check is described.



There are two paths (orange and blue) relevant for Step 1. The following expressions can be built following these two paths

Orange: $(R(1-00) \& A) \& R(1-00) \& R(1-9) \& R(1-00)$

Blue: $(R(1-00) \& A) \& R(1-00) \& R(10-00) \& R(1-00)$

Thus, the resulting expression for Step 1 is

$(R(1-00) \& A) \& R(1-00) \& R(1-9) \& R(1-00) \mid (R(1-00) \& A) \& R(1-00) \& R(10-00) \& R(1-00)$,

in which the manufacturing extended effectivity of the part $(R(1-00) \& A)$ is fully contained.

There is only one path that is relevant for Step 2 yielding the following expression

$(R(1-00) \& A) \& R(8-00) \& R(8-00) \& R(1-00)$

The cut built with expressions computed from expressions of Step 1 and Step 2 is logically equal to $(R(1-7) \& A)$

This is a guideline how to implement the User Exit that is used by acceptance tests.

Creating the UserExit DLL

Create Project

In Microsoft Developer Studio create a new MFC DLL Project. In the following it is assumed that this project is named MyUserExit. You may leave the default settings (Regular DLL using shared MFC DLL, No automation) of the project wizard.

Create Implementation File

Create a header file and an implementation file to implement the necessary interface method Execute.

The exemplary implementation of the method Execute uses the XCMCCheckAcceptanceTest class to show the usage of the exposed interface of the extended effectivity parser. The interface methods of the parser are written bold.

The error handling is reduced to minimum, in order to improve readability.

XMyUserExitAPI.h

```
#include "stdafx.h"
class XUserExitServices;

extern "C" HRESULT __stdcall Execute(XUserExitServices* pUEServices);
```

XMyUserExitAPI.cpp

```
#include "XMyUserExitAPI.h"
#include "XUserExitServices.h"
HRESULT __stdcall Execute(XUserExitServices* pUEServices)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
    if ( !pUEServices )
        return E_POINTER;
    XCMCCheckAcceptanceTest check(pUEServices);
    return check.Check();
}
```

Export the method Execute in the **MyUserExit.def** file.

MyUserExit.def

```
; MyUserExit.def : Declares the module parameters for the DLL.
LIBRARY      "MyUserExit"

EXPORTS
    Execute
```

Build Project

Build a Unicode Release version of the DLL.

Create User Exit Key in the Registry

Create in registry the 'HKEY_LOCAL_MACHINE\SOFTWARE\DELMIA\IPDSEVER\UserExit' key and below the string value "ManufacturingAccountabilityCheck" where the path and name of the DLL has to be entered.

It shouldn't make any difference which path for the DLL is chosen since there are no direct dependencies to DPE libraries.

Class XCMCCheckAcceptanceTest

The exemplary implementation of the method Execute uses the XCMCCheckAcceptanceTest class to show the usage of the exposed interface of the extended effectivity parser. The interface methods of the parser are written bold.

The XCMCCheck is the base class of the XCMCCheckAcceptanceTest class.

XCMCCheck.h

```
#pragma once

#include "basedataobj.h"
#include "epserver/userexitplugin/XUserExitServices.h"

class XCMCCheck
{
public:
    XCMCCheck(XUserExitServices* pUEServices);
    virtual ~XCMCCheck(void);

    virtual HRESULT Check(void) = 0;

protected:
    HRESULT GetServerObject(XUserExitServices::OBJECTTYPE enumObjectType,
        REFIID refIID,
        CComQIPtr<IEP_BaseDataObject>& ptrIEPBaseDataObject );

    HRESULT GetObjectById(    const CComBSTR& bstrObjectId,
        CComQIPtr<IEP_BaseDataObject>& ptrIEPBaseDataObject);

    HRESULT GetObjectByUUID(const CComBSTR& bstrUUID,
        const CComBSTR& bstrTypeName,
        CComQIPtr<IEP_BaseDataObject>& ptrIEPBaseDataObject);

    void GetAttributeValue(CComVariant& varAttributeValue);
    void GetAttributeName(CComBSTR& bstrAttributeName);
    void GetObjectId(CComBSTR& bstrObjectId);

    void WriteErrorMsg(const CComBSTR& bstrTypeName);

private:
    XUserExitServices* m_pUEServices;
};
```

XCMCCheck.cpp

```
#include "StdAfx.h"
#include "xcmccheck.h"
#include "ObjectQuery.h"

XCMCCheck::XCMCCheck(XUserExitServices* pUEServices) :
    m_pUEServices(pUEServices)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
}
```



```

XMCCheck::~XMCCheck(void)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
}

void XMCCheck::GetAttributeValue(CComVariant& varAttributeValue)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( m_pUEServices )
        m_pUEServices->GetAttributeValue(&varAttributeValue);
}

void XMCCheck::GetAttributeName(CComBSTR& bstrAttributeName)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( m_pUEServices )
        m_pUEServices->GetAttributeName(&bstrAttributeName);
}

void XMCCheck::GetObjectId(CComBSTR& bstrObjectId)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( m_pUEServices )
        m_pUEServices->GetObjectId(&bstrObjectId);
}

HRESULT XMCCheck::GetObjectById(
    const CComBSTR& bstrObjectId,
    CComQIPtr<IEP_BaseDataObject>& ptrIEPBaseDataObject)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( !m_pUEServices )
        return E_POINTER;

    CComPtr<IEP_Query> ptrIEP_Query;
    HRESULT hr = m_pUEServices->GetServerObject(
        XUserExitServices::OT_QUERY, __uuidof(IEP_Query), (void**)&ptrIEP_Query );
    if ( S_OK != hr )
        return hr;

    CComPtr<IUnknown> pUnk;
    hr = ptrIEP_Query->GetObjectById(bstrObjectId, &pUnk);
    ptrIEPBaseDataObject = pUnk;

    return hr;
}

HRESULT XMCCheck::GetObjectByUUID(
    const CComBSTR& bstrUUID,
    const CComBSTR& bstrTypeName,
    CComQIPtr<IEP_BaseDataObject>&
    ptrIEPBaseDataObject)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( !m_pUEServices )
        return E_POINTER;

    CComPtr<IEP_Query> ptrIEP_Query;
    HRESULT hr = m_pUEServices->GetServerObject(
        XUserExitServices::OT_QUERY, __uuidof(IEP_Query), (void**)&ptrIEP_Query );
    if ( S_OK != hr )
        return hr;
}

```

```

        CComPtr<IUnknown> pUnk;
        hr = ptrIEP_Query->GetObjectByUUID(bstrUUID, bstrTypeName, &pUnk);
        ptrIEPBaseDataObject = pUnk;

        return hr;
    }

HRESULT XCMCCheck::GetServerObject(
    XUserExitServices::OBJECTTYPE enumObjectType,
    REFIID refIID,
    CComQIPtr<IEP_BaseDataObject>& ptrIEPBaseDataObject )
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( !m_pUEServices )
        return E_POINTER;

    return m_pUEServices->GetServerObject(
        enumObjectType, refIID, (void**)&ptrIEPBaseDataObject );
}

void XCMCCheck::WriteErrorMsg(const CComBSTR& bstrTypeName)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());

    if ( m_pUEServices )
        m_pUEServices->WriteErrorMsg(bstrTypeName);
}

```

XCMCCheckAcceptanceTest.h

```

#pragma once
#include "xmccheck.h"

#include "epfilter2.h"

#include <string>
#include <map>
#include <vector>

class XUserExitServices;

class XCMCCheckAcceptanceTest : public XCMCCheck
{
public:
    XCMCCheckAcceptanceTest(XUserExitServices* pUEServices);
    virtual ~XCMCCheckAcceptanceTest();

    virtual HRESULT Check();

private:
    /**
     * Return the map of products ids (as key) and engineering extended effectivities
     * Products are separated by ',' and the engineering extended effectivities by '@'
     */
    void GetProducts(
        const std::wstring& strExternalData,
        std::map<std::wstring, std::wstring>& mapProducts);

    /** Split the string, wherein single parts are separated by comma */
    void Split(
        const std::wstring& strExternalData,
        std::vector<std::wstring>& vectorProducts);

    /** Set filter */
    void SetFilter(
        const CComQIPtr<IEP_BaseDataObject>& ptrProject,

```

```

        const CComQIPtr<IEP_BaseDataObject>& ptrActionProxy);

/** Perform manufacturing accountability check for a given part */
bool CheckManufacturingAccountability(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProduct,
    const CComBSTR& bstrExtendedEffectivity,
    CComBSTR& bstrExtendedEffectivityResult);

/**
 * Perform engineering accountability check for a given part and
 * an engineering extended effectivity
 */
bool CheckEngineeringAccountability(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProduct,
    const CComBSTR& bstrExtendedEffectivity,
    const CComBSTR& bstrEngineeringEffectivity,
    const CComBSTR& bstrManufacturingCheckEffectivity);

/**
 * Iterates over relations Nodes and for all paths from a given process
 * to the top level node in the process structure
 * compute the extended effectivity and store in the vector.
 */
void ComputeEffectivityOverNodes(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProcess,
    std::vector<CComBSTR>& vectorPaths,
    const CComBSTR& bstrCurrentEffectivity);

/** Compute the extended effectivity over relation source objects */
void ComputeEffectivityOverSourceObjects(
    CComQIPtr<IEnumBaseDataObject> ptrRelations,
    const CComQIPtr<IEPFilter>& ptrParser,
    std::vector<CComBSTR>& vectorPaths,
    const CComBSTR& bstrCurrentEffectivity);

/**
 * Removes left and right whitespaces
 * - whitespaces: (0x09 - 0x0D or 0x20)
 */
std::wstring& Trim (std::wstring& strIn);
};

```

XCMCCheckAcceptanceTest.cpp

```

#include "StdAfx.h"
#include "xmccheckacceptancetest.h"

#include "relation.h"
#include "epserver/userexitplugin/XUserExitServices.h"
#include "dpf/eperror/comretcodes.h"

#include <algorithm>
#include <exception>

XCMCCheckAcceptanceTest::XCMCCheckAcceptanceTest(XUserExitServices* pUEServices) :
    XCMCCheck(pUEServices)
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
}

XCMCCheckAcceptanceTest::~XCMCCheckAcceptanceTest()
{
    AFX_MANAGE_STATE(AfxGetStaticModuleState());
}

HRESULT XCMCCheckAcceptanceTest::Check()

```

```

{
    try
    {
        AFX_MANAGE_STATE(AfxGetStaticModuleState());

        CComBSTR bstrAttributeName;
        GetAttributeName(bstrAttributeName);

        // This DLL should only be called when attribute 'state' of
        // class XDOActionProxy is set
        if ( bstrAttributeName == CComBSTR(L"state"))
        {
            CComVariant varActionState;
            GetAttributeValue(varActionState);
            CComBSTR bstrActionState(varActionState.bstrVal);
            if ( bstrActionState == L"released" )
            {
                // Get action proxy object
                CComBSTR bstrActionProxyId;
                GetObjectId(bstrActionProxyId);
                CComQIPtr<IEP_BaseDataObject> ptrActionProxy;
                HRESULT hr = GetObjectById(bstrActionProxyId, ptrActionProxy);
                if ( S_OK != hr )
                    throw std::runtime_error("Internal runtime error");

                // Get attribute 'externaldata'
                // (UUID of product to be checked) of action proxy.
                CComVariant varExternalData;
                hr = ptrActionProxy->GetAttribute(
                    CComBSTR(L"externaldata"),
                    &varExternalData);
                if ( S_OK != hr )
                    throw std::runtime_error("Internal runtime error");

                // Get project
                CComVariant varProject;
                hr = ptrActionProxy->GetAttribute(
                    CComBSTR(L"ergoproject"),
                    &varProject);
                if ( S_OK != hr )
                    throw std::runtime_error("Internal runtime error");

                CComQIPtr<IEP_BaseDataObject> ptrProject (varProject.punkVal);

                SetFilter(ptrProject, ptrActionProxy);

                CComQIPtr<IEP_Filter> ptrParser = ptrProject;
                if (! ptrParser )
                    throw std::runtime_error("Internal runtime error");

                std::map<std::wstring, std::wstring> mapProducts;
                GetProducts(varExternalData.bstrVal, mapProducts);
                for(std::map<std::wstring, std::wstring>::const_iterator iter =
                    mapProducts.begin();
                    iter != mapProducts.end();
                    ++iter)
                {
                    // iterate through all parts
                    std::wstring strId = iter->first;
                    std::wstring strEffectivity = iter->second;

                    // get a part
                    CComQIPtr<IEP_BaseDataObject> ptrProduct;
                    hr = GetObjectByUUID(CComBSTR(
                        strId.c_str()),
                        CComBSTR(L"ergocompproductdefault"),
                        ptrProduct);
                    if ( S_OK != hr )

```

```

        return hr;

        CComVariant varExtendedEffectivity;
        hr = ptrProduct->GetAttribute(
            CComBSTR(L"extendedeffectivity"),
            &varExtendedEffectivity);
        if ( S_OK != hr )
            throw std::runtime_error("Internal runtime error");

        CComQIPtr<IEPRelation> ptrProductRel = ptrProduct;
        if (! ptrProductRel )
            throw std::runtime_error("Internal runtime error");

        // perform manufacturing accountability check
        CComBSTR bstrExtendedEffectivityResult;
        if (!CheckManufacturingAccountability(
            varProject.punkVal,
            ptrProduct,
            varExtendedEffectivity.bstrVal,
            bstrExtendedEffectivityResult))

            return E_USER_EXIT_FAILED;

        // perform engineering accountability check
        if (!CheckEngineeringAccountability(
            varProject.punkVal,
            ptrProduct,
            varExtendedEffectivity.bstrVal,
            CComBSTR(strEffectivity.c_str()),
            bstrExtendedEffectivityResult))

            return E_USER_EXIT_FAILED;

    }

    }
    return S_OK;
}
catch(std::exception&)
{
    return E_FAIL;
}
}

void XCMCCheckAcceptanceTest::SetFilter(
    const CComQIPtr<IEP_BaseDataObject>& ptrProject,
    const CComQIPtr<IEP_BaseDataObject>& ptrActionProxy)
{
    HRESULT hr = ptrProject->SetAttribute(CComBSTR(L"filter_action"), CComVariant(TRUE));
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");

    CComPtr<IEnumBaseDataObject> ptrMods;
    hr = ptrActionProxy->GetChildren(CComBSTR(L"modstatement"), &ptrMods);
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");

    // take any mod to set it as filter
    CComPtr<IEP_BaseDataObject> ptrMod;
    hr = ptrMods->GetSingleNext(&ptrMod);
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");

    hr = ptrProject->SetAttribute(
        CComBSTR(L"selectedmodstatement"),
        CComVariant(ptrMod));
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");
}

```

```

}

bool XCMCCheckAcceptanceTest::CheckManufacturingAccountability(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProduct,
    const CComBSTR& bstrExtendedEffectivity,
    CComBSTR& bstrExtendedEffectivityResult)
{
    CComQIPtr<IEPRelation> ptrProductRel = ptrProduct;
    if (! ptrProductRel )
        throw std::runtime_error("Internal runtime error");

    CComQIPtr<IEnumBaseDataObject> ptrRelations;
    HRESULT hr = ptrProductRel->GetRelationshipsByNameEnum(
        TRUE,
        CComBSTR(L"proc_firstprocesses_prod_reverse"),
        &ptrRelations);

    if (FAILED(hr))
        throw std::runtime_error("Internal runtime error");

    if (S_OK == hr)
    {
        std::vector<CComBSTR> vectorPaths;
        ComputeEffectivityOverSourceObjects(
            ptrRelations,
            ptrParser,
            vectorPaths,
            bstrExtendedEffectivity);

        // create a resulting effectivity expression by connecting paths by OR
        for(std::vector<CComBSTR>::const_iterator iter = vectorPaths.begin();
            iter != vectorPaths.end();
            ++iter)
        {
            hr = ptrParser->OrExtendedEffectivities(
                bstrExtendedEffectivityResult,
                (*iter),
                &bstrExtendedEffectivityResult);
            if ( S_OK != hr )
                throw std::runtime_error("Internal runtime error");
        }

        // perform manufacturing accountability check
        BOOL bContained = FALSE;
        hr = ptrParser->ContainedInExtendedEffectivity(
            bstrExtendedEffectivity,
            bstrExtendedEffectivityResult,
            &bContained);
        if ( S_OK != hr )
            throw std::runtime_error("Internal runtime error");

        return (bContained == TRUE)? true : false;
    }
    // A relation between the product and a process ("proc_firstprocesses_prod")
    // must be established
    return false;
}

bool XCMCCheckAcceptanceTest::CheckEngineeringAccountability(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProduct,
    const CComBSTR& bstrExtendedEffectivity,
    const CComBSTR& bstrEngineeringEffectivity,
    const CComBSTR& bstrManufacturingCheckEffectivity)
{
    CComQIPtr<IEPRelation> ptrProductRel = ptrProduct;

```

```

    if (! ptrProductRel )
        throw std::runtime_error("Internal runtime error");

    CComQIPtr<IEnumBaseDataObject> ptrRelations;
    HRESULT hr = ptrProductRel->GetRelationshipsByNameEnum(
        TRUE,
        CComBSTR(L"proc_removes_prod_reverse"),
        &ptrRelations);
    if (FAILED(hr))
        throw std::runtime_error("Internal runtime error");

    CComBSTR bstrEngineeringCheckEffectivity;
    if (S_OK == hr)
    {
        std::vector<CComBSTR> vectorPaths;
        ComputeEffectivityOverSourceObjects(
            ptrRelations,
            ptrParser,
            vectorPaths,
            bstrExtendedEffectivity);

        // create a resulting effectivity expression by connecting paths by OR
        for(std::vector<CComBSTR>::const_iterator iter = vectorPaths.begin();
            iter != vectorPaths.end();
            ++iter)
        {
            hr = ptrParser->OrExtendedEffectivities(
                bstrEngineeringCheckEffectivity,
                (*iter),
                &bstrEngineeringCheckEffectivity);
            if ( S_OK != hr )
                throw std::runtime_error("Internal runtime error");
        }
    }

    CComBSTR bstrExtendedEffectivityResult;
    hr = ptrParser->CutExtendedEffectivity(
        bstrManufacturingCheckEffectivity,
        bstrEngineeringCheckEffectivity,
        &bstrExtendedEffectivityResult);
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");

    // perform engineering accountability check
    BOOL bEqual = FALSE;
    hr = ptrParser->EqualExtendedEffectivity(
        bstrExtendedEffectivityResult,
        bstrEngineeringEffectivity,
        &bEqual);
    if ( S_OK != hr )
        throw std::runtime_error("Internal runtime error");

    return (bEqual == TRUE)? true : false;
}

void XCMCCheckAcceptanceTest::ComputeEffectivityOverNodes(
    const CComQIPtr<IEPFilter>& ptrParser,
    const CComQIPtr<IEP_BaseDataObject>& ptrProcess,
    std::vector<CComBSTR>& vectorPaths,
    const CComBSTR& bstrCurrentEffectivity)
{
    CComQIPtr<IEPRelation> ptrProcessRel = ptrProcess;
    if (! ptrProcessRel )
        throw std::runtime_error("Internal runtime error");

    CComQIPtr<IEnumBaseDataObject> ptrRelations;
    HRESULT hr = ptrProcessRel->GetRelationshipsByNameEnum(
        TRUE,

```

```

        CComBSTR(L"nodes_reverse"),
        &ptrRelations);
    if (FAILED(hr))
        throw std::runtime_error("Internal runtime error");

    if(S_FALSE == hr)
        vectorPaths.push_back(bstrCurrentEffectivity);
    else if(S_OK == hr)
        ComputeEffectivityOverSourceObjects(
            ptrRelations,
            ptrParser,
            vectorPaths,
            bstrCurrentEffectivity);
    else
        throw std::runtime_error("Internal runtime error");
}

void XCMCCheckAcceptanceTest::ComputeEffectivityOverSourceObjects(
    CComQIPtr<IEnumBaseDataObject> ptrRelations,
    const CComQIPtr<IEPFilter>& ptrParser,
    std::vector<CComBSTR>& vectorPaths,
    const CComBSTR& bstrCurrentEffectivity)
{
    unsigned long ulRet;
    CComPtr<IEP_BaseDataObject> ptrRelation;
    while(ptrRelations->Next(1, &ptrRelation, &ulRet) == S_OK)
    {
        CComVariant varRelExtendedEffectivity;
        HRESULT hr = ptrRelation->GetAttribute(
            CComBSTR(L"extendedeffectivity"),
            &varRelExtendedEffectivity);
        if ( S_OK != hr )
            throw std::runtime_error("Internal runtime error");

        CComBSTR bstrEffectivity;
        hr = ptrParser->AndExtendedEffectivities(
            bstrCurrentEffectivity,
            varRelExtendedEffectivity.bstrVal,
            &bstrEffectivity);
        if ( S_OK != hr )
            throw std::runtime_error("Internal runtime error");

        CComVariant varProcess;
        hr = ptrRelation->GetAttribute(
            CComBSTR(L"relationobjectsourcesource"),
            &varProcess);
        if ( S_OK != hr )
            throw std::runtime_error("Internal runtime error");

        ComputeEffectivityOverNodes(
            ptrParser,
            varProcess.punkVal,
            vectorPaths,
            bstrEffectivity);

        ptrRelation = 0;
    }
}

void XCMCCheckAcceptanceTest::GetProducts(
    const std::wstring& strExternalData,
    std::map<std::wstring, std::wstring>& mapProducts)
{
    mapProducts.clear();

    std::vector<std::wstring> vectorProducts;
    Split(strExternalData, vectorProducts);
}

```


Exemplary Manufacturing Accountability Check

```

        for(std::vector<std::wstring>::const_iterator iter = vectorProducts.begin();
            iter != vectorProducts.end();
            ++iter)
        {
            std::wstring::size_type pos;
            pos = (*iter).find('@');
            std::wstring strKey = Trim((*iter).substr(0, pos));
            std::wstring strEff = (pos != std::wstring::npos)?
                Trim((*iter).substr(pos + 1, (*iter).size() - pos)) : L"";

            mapProducts.insert(std::make_pair(strKey, strEff));
        }
    }

void XCMCCheckAcceptanceTest::Split(
    const std::wstring& strExternalData,
    std::vector<std::wstring>& vectorProducts)
{
    vectorProducts.clear();

    for(std::wstring::const_iterator iter = strExternalData.begin(), pos;
        (pos = std::find(iter, strExternalData.end(), ',')) != strExternalData.end();
        iter = ++pos)

        vectorProducts.push_back(std::wstring(iter, pos));

    if (iter != strExternalData.end())
        vectorProducts.push_back(std::wstring(iter, strExternalData.end()));
}

std::wstring& XCMCCheckAcceptanceTest::Trim(std::wstring& strIn)
{
    while (iswspace(strIn[0]))
        strIn.erase(0, 1); // trim left

    while (iswspace(strIn[strIn.length() - 1]))
        strIn.erase(strIn.length() - 1, 1); // trim right

    return strIn;
}

```