WORK ORDER COSTING HOOK

DECEMBER 2021

FSM Program Enhancements to support WFM integration

Acknowledgement of Country

Hunter Water operates across the traditional country of the Awabakal, Birpai, Darkinjung, Wonaruah and Worimi peoples. We recognise and respect their cultural heritage, beliefs and continuing relationship with the land, and acknowledge and pay respect to Elders past, present and future.

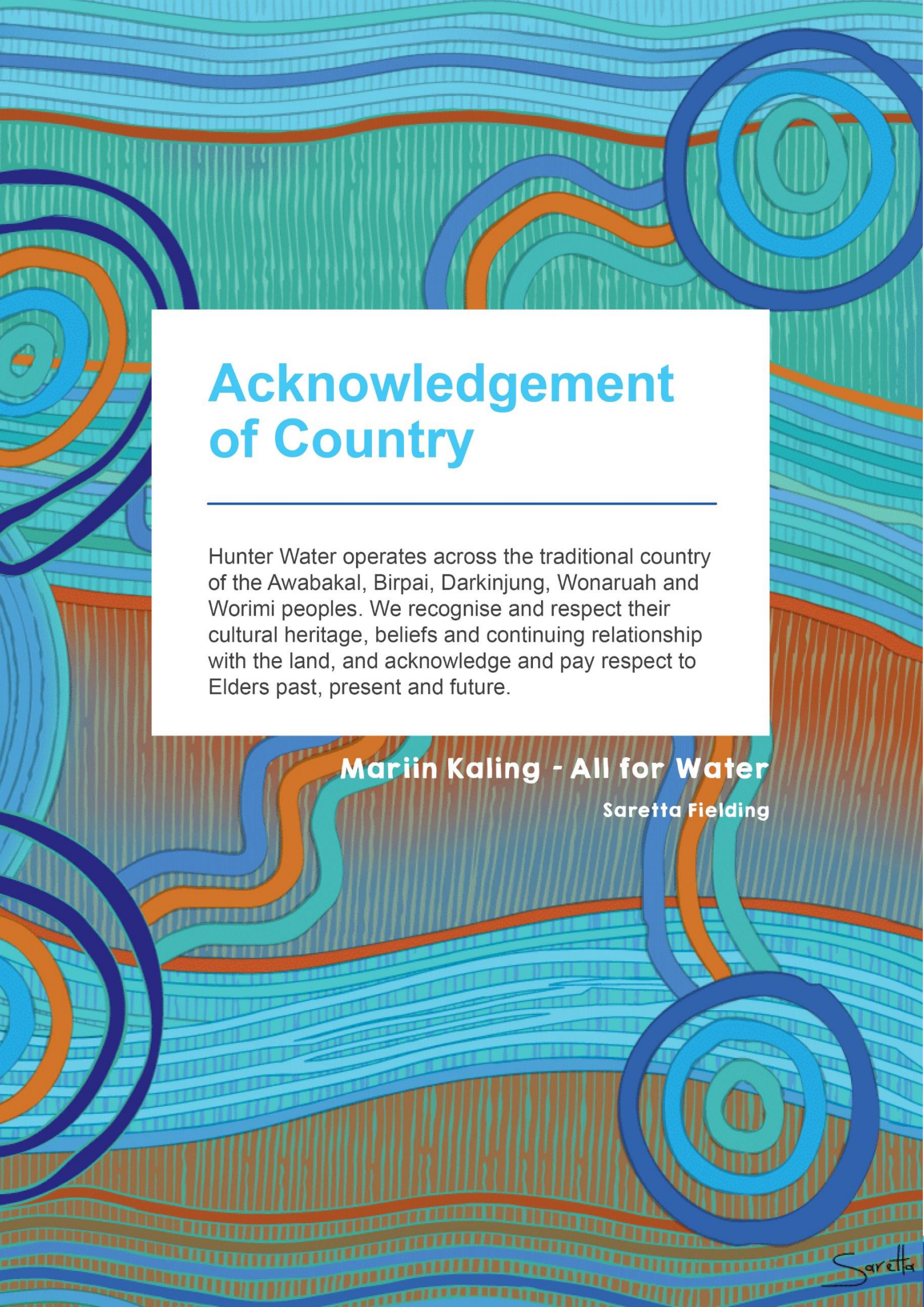


TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

Hunter Water has initiated a major program to deliver improvements by implementing Field Services Mobility across the Maintenance Services Division. An analysis of the existing work management and financial accounting process has identified gaps in an existing Ellipse customisation. If left unchanged the business rules will restrict changes to Work Orders that are intended to improve data quality and accurate accounting of cost per job.

This document outlines the changes required to support FSM. A number of options where considered and following business consultation it has been decided to make enhancements to the Ellipse Work Order Hooks. In addition, the interface - system account, used by the FSM to Ellipse will need to be configured to achieve the necessary updates to Work Orders to ensure accurate job costing.

The investigation also identified that the Hook while initially developed against the correct Service – WorkOrder, has been now identified by the Ellipse Vendor as legacy. All changes to the Hook will need to be implemented in both the Work Service and WorkOrder Service to ensure existing interfaces and connectors will continue to work consistently with the UI.

2. INTRODUCTION

2.1. Purpose

Early versions of Ellipse 8 maintained a single service operation to update Work Order details. A minor release after the initial Hunter Water Upgrade, introduced a change to the Work Order Service and Hook. The Hook applies business rules to derive the correct Work Order Account Code and help streamline accurate Job costing. The change introduced to the Work Order has left some unhandled scenarios with the move to Ellipse 9 that will materially affect the interface between FSM and Ellipse.

2.2. Scope

This specification will outline the necessary changes to the Ellipse Work Order Hooks to handle the identified scenarios to support FSM. The delivered work package will also need to include unit testing, any deployment and configuration change documentation support for UAT, Go-live and Post Go-live for a defined period as per standard Hunter Water procurement Terms & Conditions.

2.3. Audience

The document is intended to be read and understood by the Program Management, Business Owners and Service Providers component to maintain Ellipse Groovy Hooks in an Ellipse 9 environment.

2.4. Definitions and Acronyms

Term	Description
WO	Work Order
HDWB	The default Hunter Water district in Ellipse

2.5. References

ID	Description	Jira Reference
REF-01	Ellipse 8 Upgrade – WO Service Enhancements (HW20212-986 1.022)	
REF-02	FSM – WO Costing Hook Testing Analysis	

Note: - References have been attached in Appendix for Service Provider awareness

2.6. Version

ID	Description	Jira Reference
VER-01	Initial Spec and Scope of Works	
VER-02	Updated to include both Work and WorkOrder Service implementations along with additional testing requirements	

3. REQUIREMENTS

ID	Description	Jira Reference
REQ-01	**Webservice and UI business rules applied consistently** The changes to the business rules are uniformly applied and no deviation exists between	

	Webservice interaction and User Interface of Ellipse.	
REQ-02	**Upgrade Hook to apply to both WorkOrder Service and Work Service (technical requirement)** The existing hook interacts with the WorkOrder Service. In Ellipse 9 this service was identified as legacy and any new functionality or enhancements would only be applied to the WorkOrder and Work Service.	
REQ-03	**Reinstate Account Code changes when no Work Order costs exist** Currently, once the work order is created only +WKR identified resources can change the account code. The business rule implementation has been incorrectly applied when the Service was changed as it was never the intent to lock out changes when no costs exist. The original intent needs to be reinstated to support FSM processes around asset identification.	
REQ-05	**Support Retirement or Business Responsibility Changes** When a piece of equipment is retired, i.e. moved to district 0001 or the equipment has Business Responsibility changed, e.g. moved to district 9999, a warning is displayed but the Hook appears not to handle the scenario. When a WO is created and the equipment is a non HDBW district a warning appears but it lets you proceed.	

4. PROCESS OVERVIEW

4.1. Current Business Practice

4.1.1. Overview

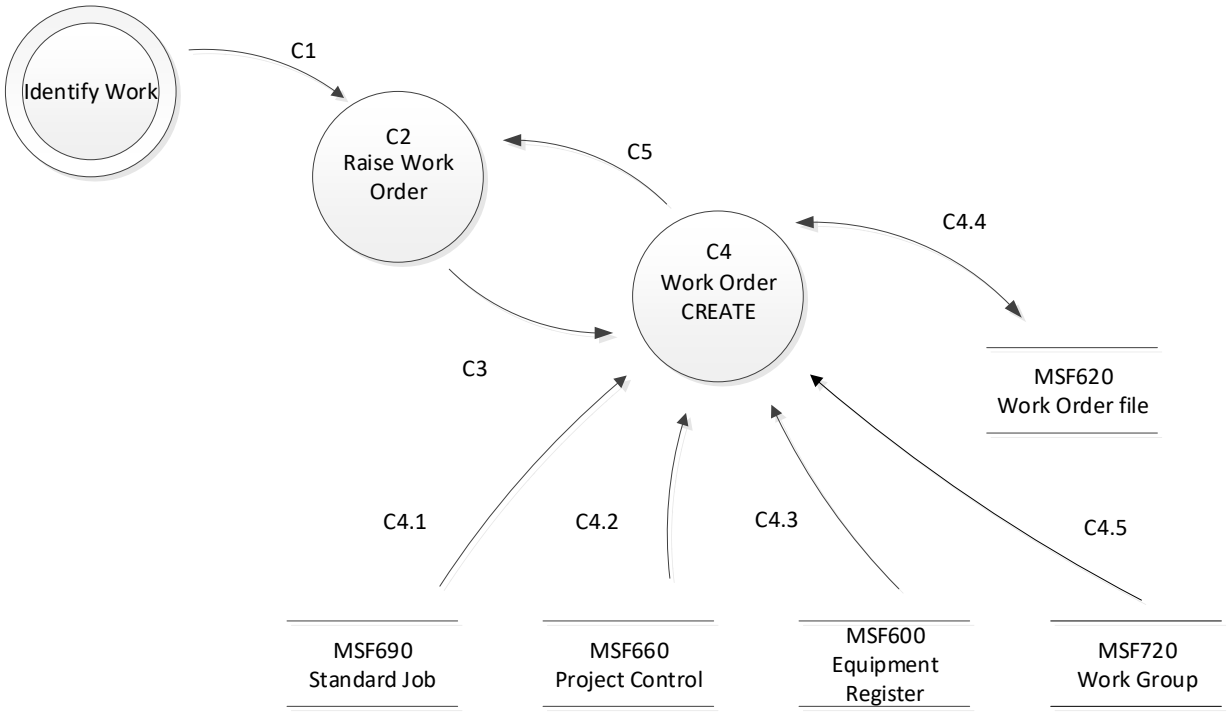
A change is required to the existing CREATE and MODIFY work order processes to correct issues arising from the Service Operation changes associated with Work Orders that were introduced as a minor upgrade.

Historically, it has been assumed that cost allocations have been correctly applied and while this has been the case some scenarios a number of gaps have been identified. Furthermore, the proposed interface between Ellipse and FSM has increased the urgency to close these process gaps

Hunter Water Corporation have previously implemented changes to business logic in the CREATE and MODIFY Work Order processes as part of the Work to Work Order Project, implemented in 2009. These changes were then further extended as part of the 2014 Ellipse 5 to Ellipse 8 upgrade.

The hook the following logic in concert with Standard Ellipse function. **Error! Reference source not found.** and **Error! Reference source not found.** below provide a decision tree representation of how the Account Code on a Work Order is determined.

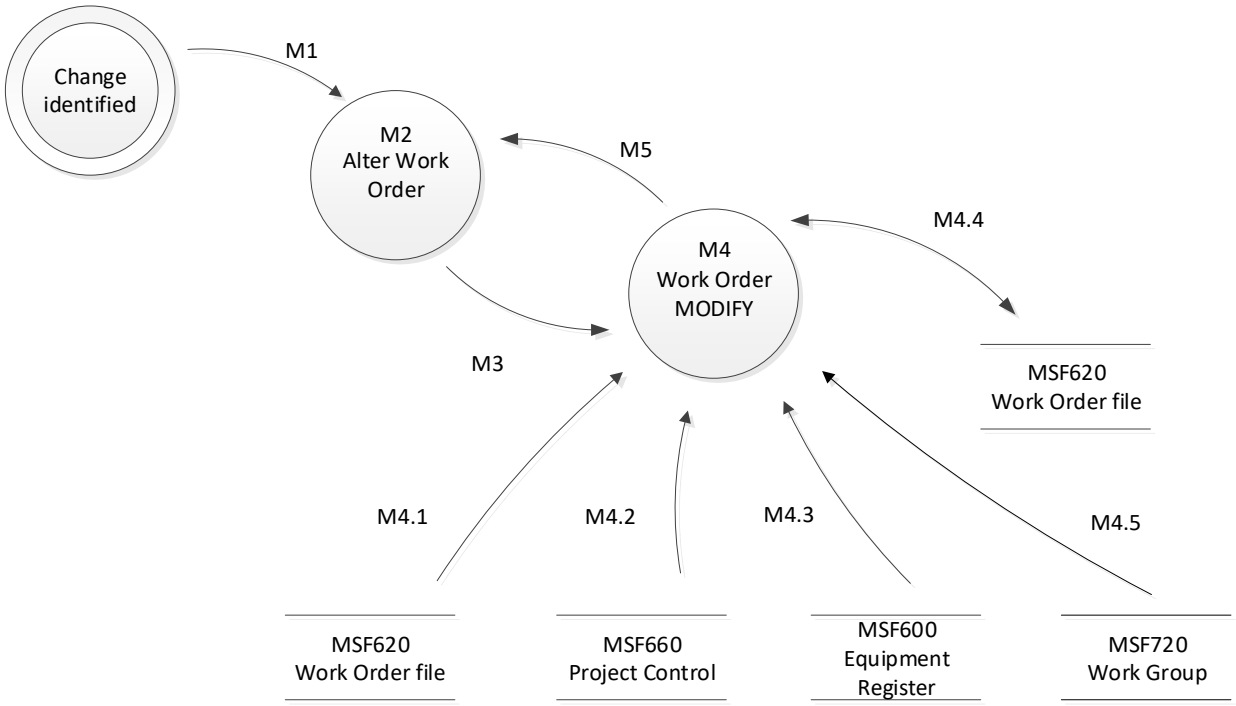
4.1.2. CREATE Process Data Flow



Custom Hook applied at *C4. Work Order CREATE* includes:

- Mandate Equipment & Work Group
- Account Code determination

4.1.3. MODIFY Process Data Flow



Custom Hook applied at *M4 Work Order MODIFY* includes:

- Mandate Equipment & Work Group
- Limit change of Project, Parent Work Order and Account Code

4.1.4. Create Business Rules

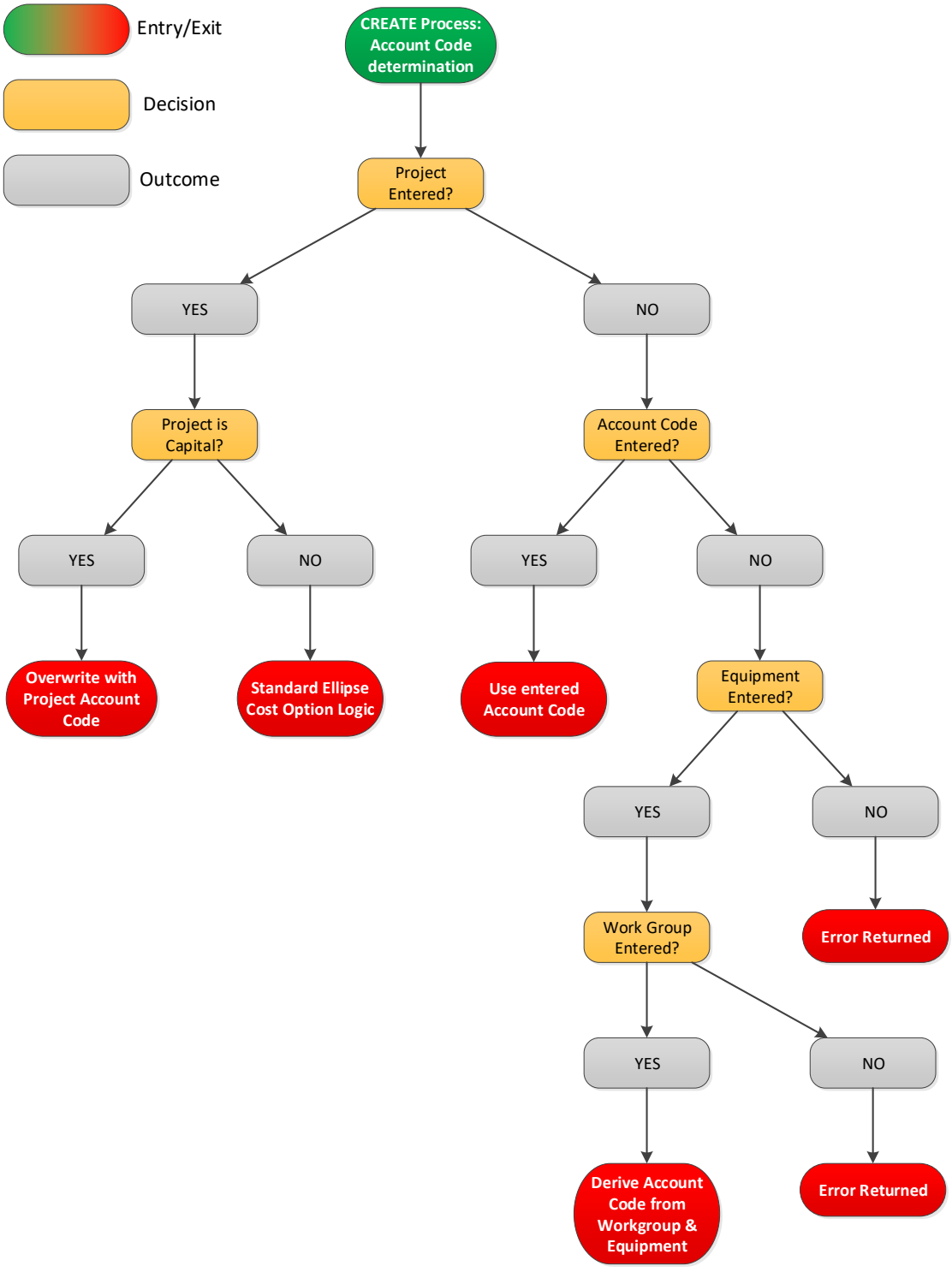


Figure 1: Current CREATE Business Rules - Account Code decision tree

4.1.5. Modify Business Rules

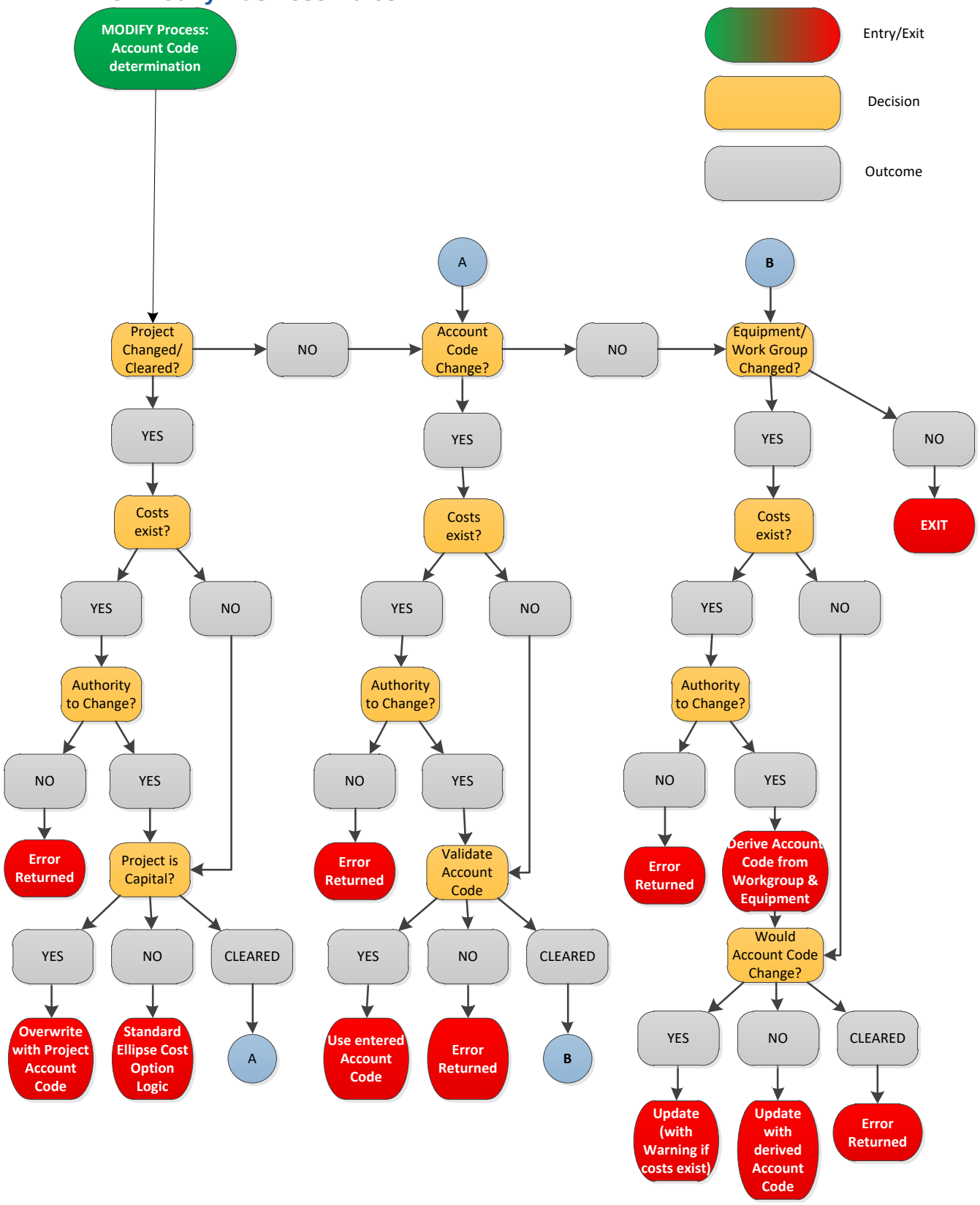


Figure 2: Current MODIFY Business Rules - Account Code decision tree

The modify process handles the following scenarios: -

INITIAL STATE

Scenario	Project No	Account Code	Equipment No & Work Group
1	Exists	Exists	Mandatory
2	BLANK	Exists	Mandatory

PROCESS LOGIC – Project Number Change

If no costs exist applies these changes

Initial State	Cleared	New
BLANK	Do nothing	If costs exist, • verify authority, • ensure MSB664 is triggered.
Exists	If costs exist, • verify authority, • ensure MSB664 is triggered.	If costs exist, • verify authority, • ensure MSB664 is triggered.

PROCESS LOGIC – Account Code Change

Initial State	Cleared	New
Exists (with project no.)	If no costs exist apply account code from project. If costs exist, • verify authority, • ensure MSB664 is triggered.	If no costs exist over write account code from project. If costs exist, • verify authority, • ensure MSB664 is triggered.
Exists (no project no.)	If no costs exist determine account code from Equipment/Work Group. If costs exist,	If no costs exist apply account code. If costs exist,

	• verify authority[1],	• verify authority[2],

PROCESS LOGIC – Equipment Number & Work Group Change

Note: A new default account will only be applied if the existing account code is cleared else the logic will check and return warning ONLY that default account should be changed.

Initial State	Determined Account Code is Same	Determined Account Code Change
Mandatory	Do nothing	If costs exist, • verify authority[3],

[1] MANUAL PROCEDURE REQUIRED – Authorised users will be required to follow a procedure to determine whether existing costs will need to be reallocated to the new cost centre.

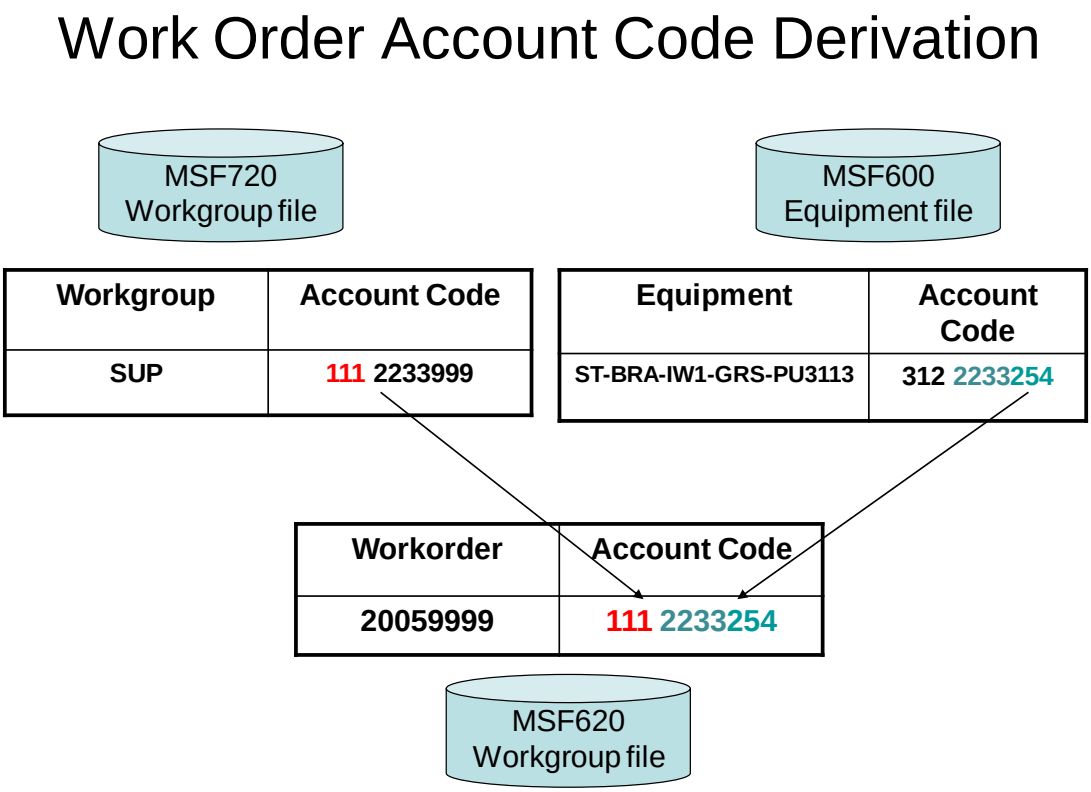
[2] MANUAL PROCEDURE REQUIRED – Authorised users will be required to follow a procedure to determine whether existing costs will need to be reallocated to the new cost centre.

[3] MANUAL PROCEDURE REQUIRED – Authorised users will be required to follow a procedure to determine whether existing costs will need to be reallocated to the new cost centre.

4.2. Account Derivation Overview

Work Order account code will be derived as follows: -

- For the Workgroup assigned to the work order extract the ORG UNIT from the Account Code,
- For the Equipment assigned to the work order extract the ACTIVITY from the Account Code,
- Construct a new account code for the Work Order by combining the Workgroup ORG UNIT and the Equipment ACTIVITY.



4.3. Proposed Business Practice

4.3.1. Overview

The proposed changes should not materially affect the basic Business Process associated with a work order CREATE or MODIFY but will impact the underlining technical implementation.

Additionally, configurations should remain unchanged and have been summarised below for reference: -

1. Workgroups will still need a default account code where Segment 1 - ORG UNIT reflects that of the Workgroup,

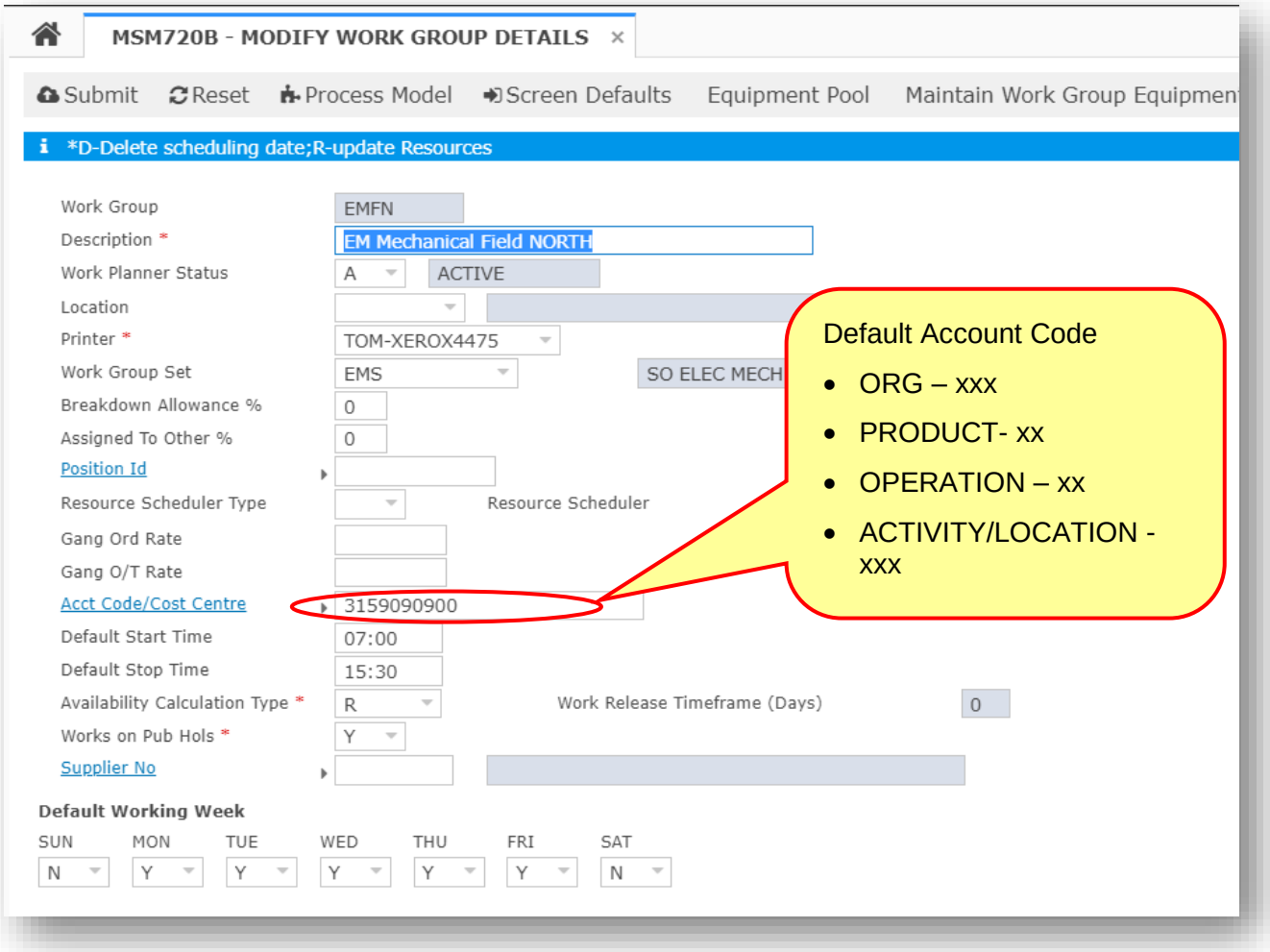


Figure 3: MSE720 Option2 - Modify Work Group Details (MSM720B)

2. Equipment will still need a default account code where Segment 2-4 reflect that of the Product/Operation/Activity/Location of the Equipment,

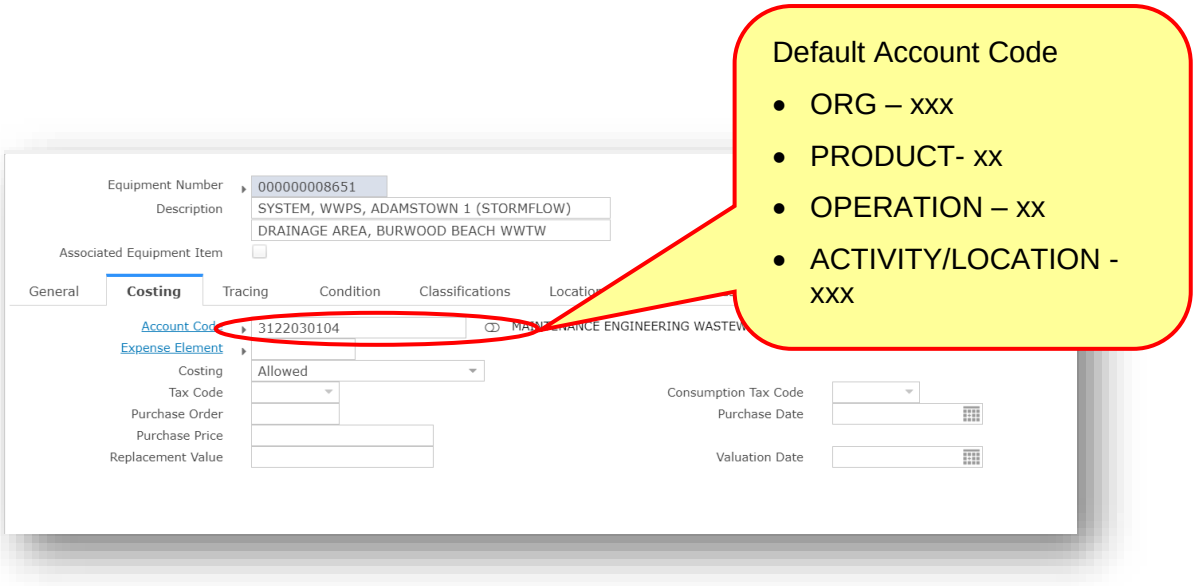


Figure 4: MSE600 Equipment Detail - Costing Tab

3. Limiting Account Code changes to authorised users once transactions (Costs) exist against a Work Order will still be recorded against the +WKR Table File.

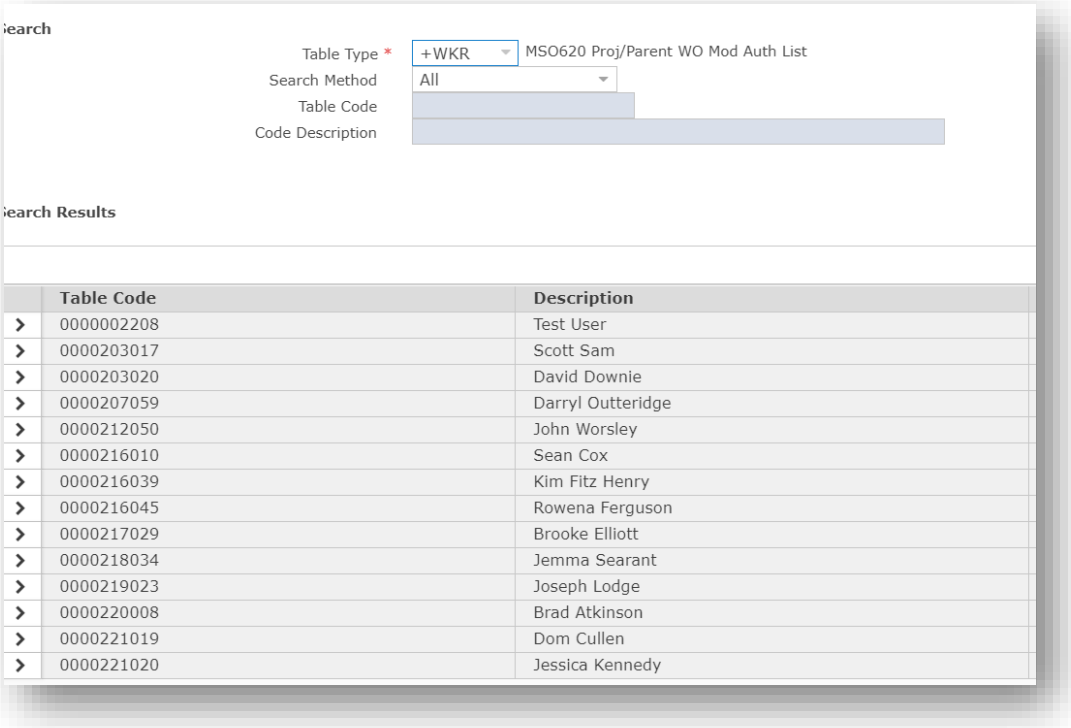


Table Code	Description
0000002208	Test User
0000203017	Scott Sam
0000203020	David Downie
0000207059	Darryl Outteridge
0000212050	John Worsley
0000216010	Sean Cox
0000216039	Kim Fitz Henry
0000216045	Rowena Ferguson
0000217029	Brooke Elliott
0000218034	Jemma Searant
0000219023	Joseph Lodge
0000220008	Brad Atkinson
0000221019	Dom Cullen
0000221020	Jessica Kennedy

Figure 5: MSE010 Table File - +WKR WO Mod Auth List

4. The identified Service and operations impacted by the proposed change are highlighted in Figure 6: WorkService Operations and Figure 7: WorkOrderService Operations below.

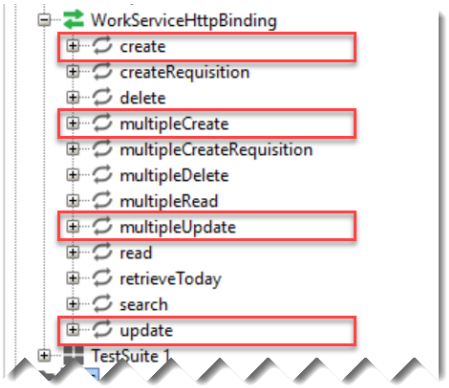


Figure 6: WorkService Operations

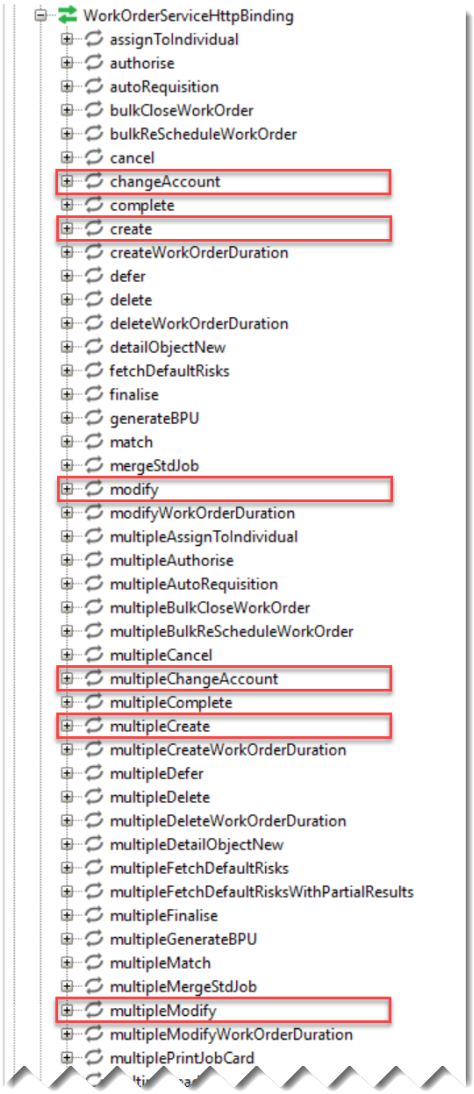


Figure 7: WorkOrderService Operations

An additional requirement identifies changes to the existing logic applied in the hook. This change is: -

1) Support Retirement or Business Responsibility Changes

When a piece of equipment is retired, i.e. moved to district 0001 or the equipment has Business Responsibility changed, e.g. moved to district 9999, a warning is displayed but the Hook appears not to handle the scenario. When a WO is created and the equipment is a non HDBW district a warning appears but it lets you proceed.

4.3.2. CREATE Process Data Flow

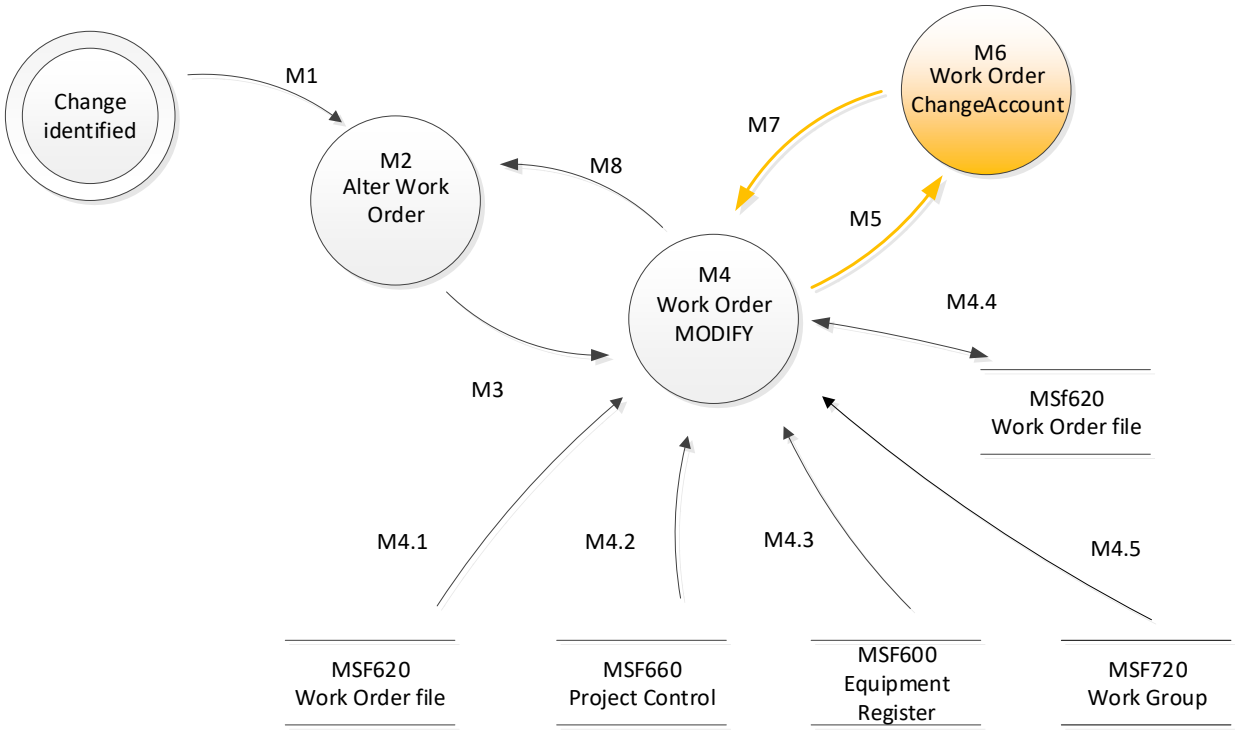
The CREATE Process will need to support changes via both the user interface and service layer.

No changes have been identified against the existing process only changes to the implementation have been identified.

A new hook using Work.create service to be developed and existing business rules applied to achieve the desired outcome.

The scope of work is to transfer the hook logic to the new Work Service.

4.3.3. MODIFY Process Data Flow



The MODIFY Process will need to support changes via both the user interface and service layer.

This process may need to hook business logic to both the Work.modify and Work.changeAccount services to achieve the desired outcome via a single transaction as well as apply changes to Work.changeAccount for standard Ellipse process support.

The scope of work will include transferring the Hook against the new Work Service.

4.3.4. Create Business Rules

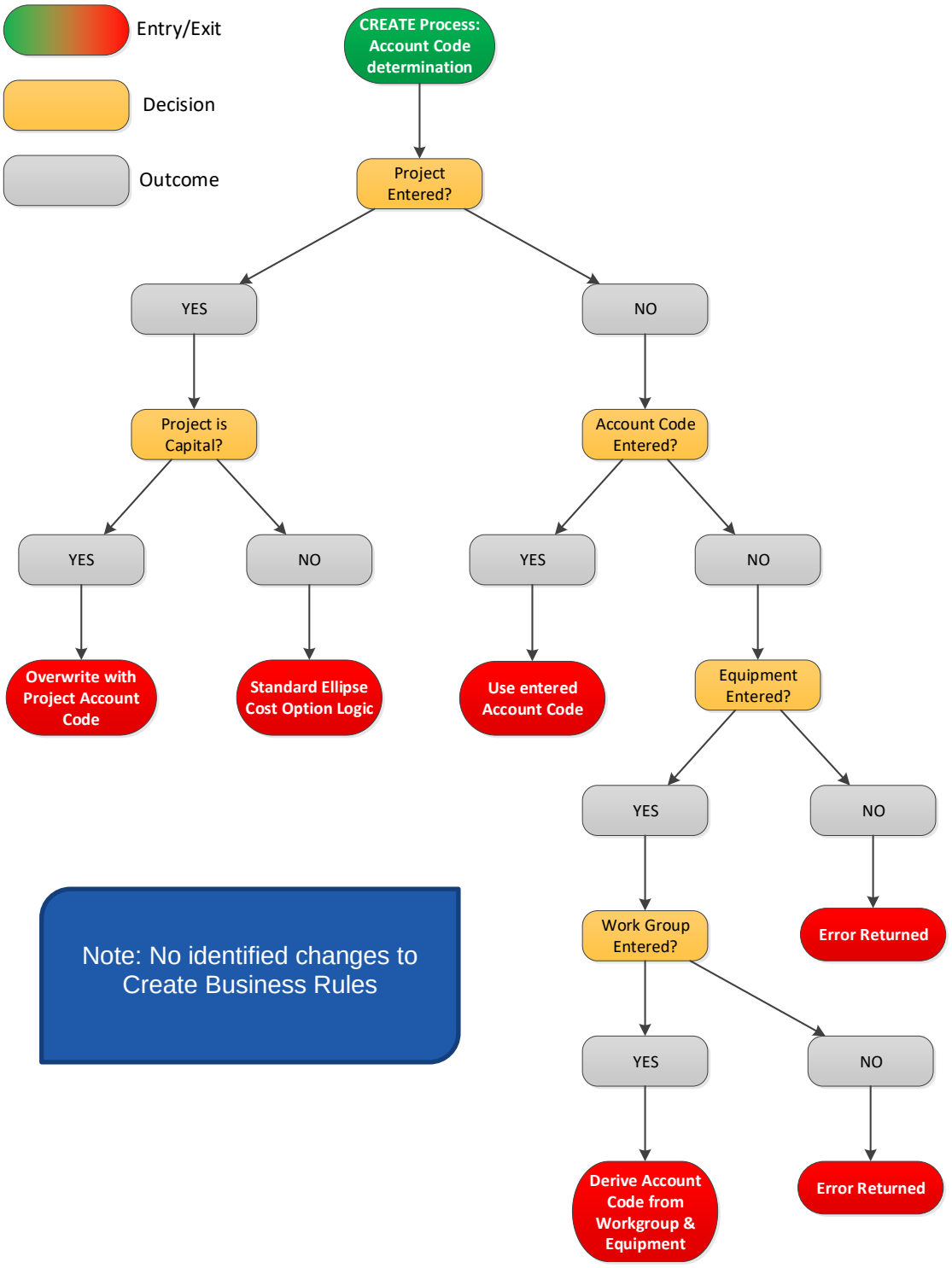


Figure 8: Proposed CREATE Business Rules - Account Code decision tree

4.3.5. Modify Business Rules

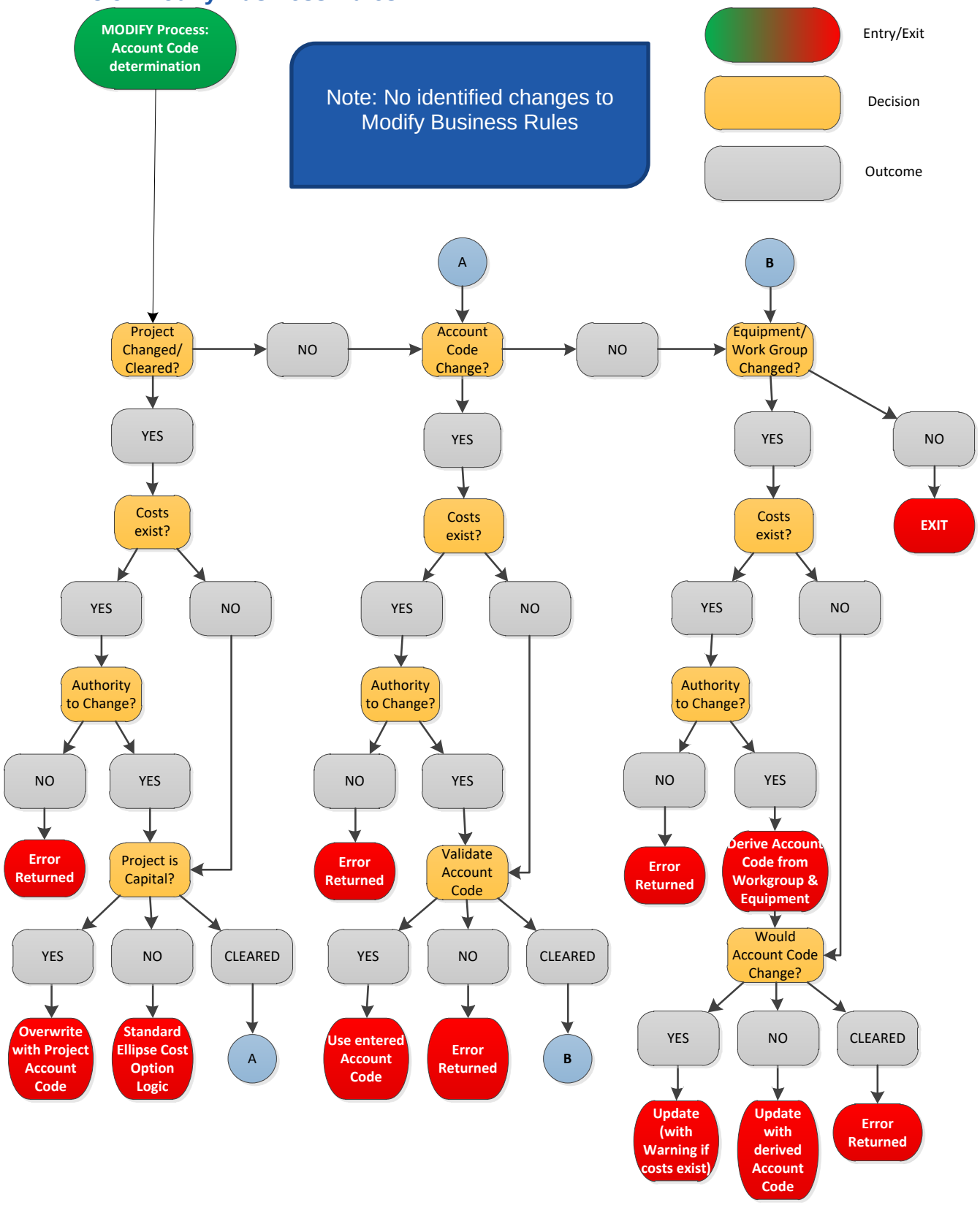


Figure 9: Proposed MODIFY Business Rules - Account Code decision tree

5. ASSUMPTION AND LIMITATIONS OF CURRENT PROCESS & FUNCTIONALITY

1) Assume that the logic can be captured against the Work webservice call and not in the Applications

2) Any creation of work orders not using webservices will need to be manually updated to include this logic. (Outside scope of this requirement)

3) All non-project Work Orders can have their account code changed at ANYTIME up to the first transaction using business rules applied by the hook. After the first transaction an authorised user will be required to process the change online.

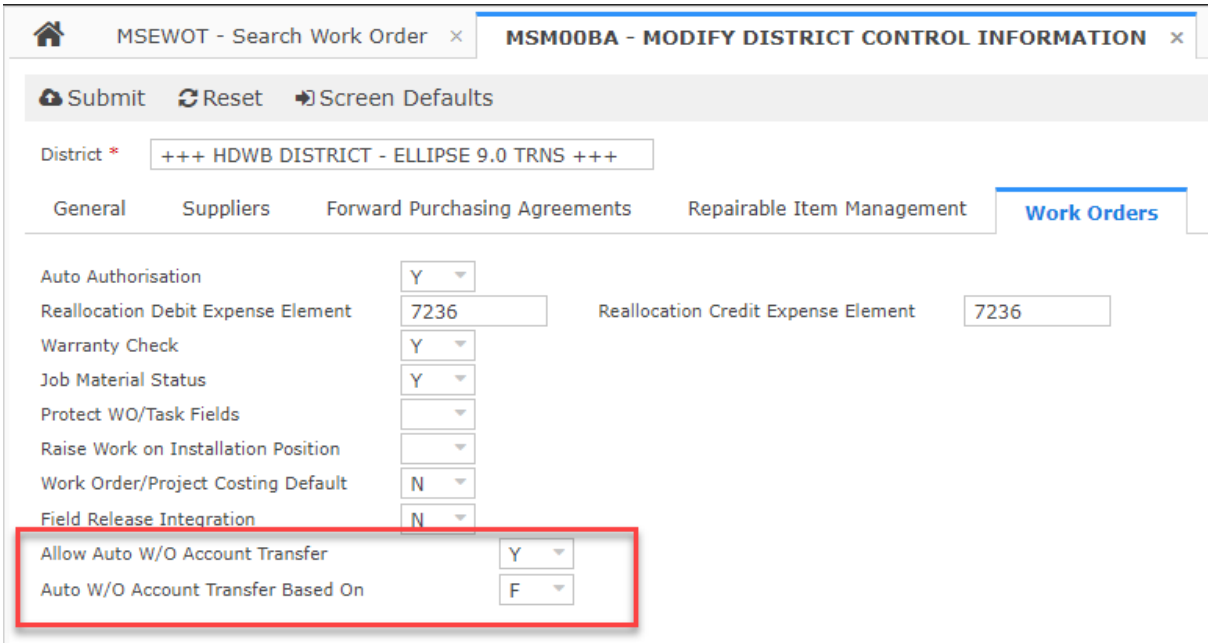
4) Authorised users are recorded and maintained in the +WKR table file.

6. TECHNICAL DETAILS

6.1. System Configuration

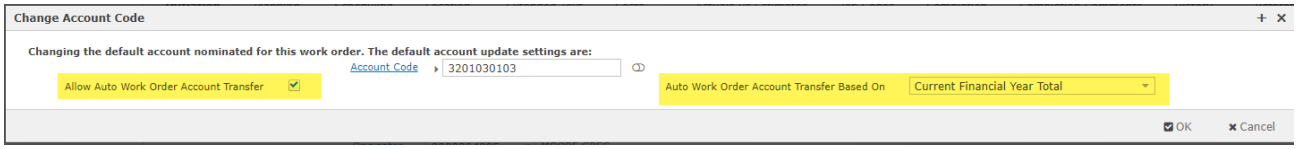
Configuration Settings

Hunter Water has the following relevant system configuration settings in ELL9PRD MSO00BA that developers should be aware of: -



Allow Auto Work Order Account Transfer – set as '**Yes**' [**Y**] (see ticked below), and

Auto W/O Account Transfer Based On – set as '**Current Financial Year Total**' [**F**]



The result of the settings is visible when attempting to change an account code in the work Order application MSEWOT or via a web service call. The above dialog is displayed in the application and set as per the system settings in MSO00BA.

7. ACCEPTANCE CRITERIA

The acceptance criteria have been developed using the below assumptions:

1) Authorised account code change user will use the Ellipse UI or Smart Excel (no connectors have been identified in these business process)
2) The Work Order Closure connector is used solely by EMM to closure work orders. These users are NOT a subset of "Authorised Account Code change user".
3) Non-authorised account code change users will also use Ellipse UI or Smart Excel to change Work Orders and both technical implementations will need to be tested.
4) Where smart excel is being used it is assumed a user may utilise the Multi* Operations of each service and thus they will need to be tested separately.

7.1. Authorised Account Code change user

7.1.1. Create Work Order User Story – Ellipse UI

1. Use MSEWOT to create work order
   a. Create without a project number
      i. Validate Workgroup and Equipment are mandatory,
      ii. Validate derived Account Code is correct.
   b. Create with a project number
      i. Validate Workgroup and Equipment are mandatory,
      ii. Validate derived Account Code is correct.

7.1.2. Modify Work Order User Story – Ellipse UI

1. Use MSEWOT to modify work order
   a. Modify without a project number
      i. Validate Workgroup and Equipment are mandatory,
      ii. Validate costs are checked
      iii. Validate the default flags on Change Account Dialog are set
      iv. Validate +WKR user is checked and transaction proceeds
      v. Validate derived Account Code is correct.
   b. Modify with a project number
      i. Validate Workgroup and Equipment are mandatory,

ii. Validate costs are checked
iii. Validate the default flags on Change Account Dialog are set
iv. Validate +WKR user is checked and transaction proceeds
v. Validate derived Account Code is correct.

7.1.3. Create Work Order User Story – Webservice (Work Service)

1. Use MSEWOT to create work order
    a. Create without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.
    b. Create with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.

7.1.4. Create Work Order User Story – Webservice (WorkOrder Service)

2. Use MSEWOT to create work order
    c. Create without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.
    d. Create with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.

7.1.5. Modify Work Order User Story – Webservice (Work Service)

1. Use MSEWOT to modify work order
    a. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction proceeds
        v. Validate derived Account Code is correct.
    b. Modify with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction proceeds
        v. Validate derived Account Code is correct.

7.1.6. Modify Work Order User Story – Webservice (WorkOrder Service)

2. Use MSEWOT to modify work order
    c. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction proceeds
        v. Validate derived Account Code is correct.
    d. Modify with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction proceeds
        v. Validate derived Account Code is correct.

7.1.7. Create Work Order (Multi) User Story – Smart Excel (Work Service)

Apply 7.1.1 using 5 or more records

7.1.8. Create Work Order (Multi) User Story – Smart Excel (WorkOrder Service)

Apply 7.1.1 using 5 or more records

7.1.9. Modify Work Order (Multi) User Story – Smart Excel (Work Service)

Apply 7.1.2 using 5 or more records

7.1.10. Modify Work Order (Multi) User Story – Smart Excel (WorkOrder Service)

Apply 7.1.2 using 5 or more records

7.2. Non-authorised Account Code change user

7.2.1. Create Work Order User Story – Ellipse UI

1. Use MSEWOT to create work order
    a. Create without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.
    b. Create with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.

7.2.2. Modify Work Order User Story – Ellipse UI

1. Use MSEWOT to modify work order
    a. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.
    b. Modify with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.

7.2.3. Create Work Order User Story – Webservice (Work Service)

1. Use MSEWOT to create work order
    a. Create without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.
    b. Create with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.

7.2.4. Create Work Order User Story – Webservice (WorkOrder Service)

2. Use MSEWOT to create work order
    c. Create without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.
    d. Create with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate derived Account Code is correct.

7.2.5. Modify Work Order User Story – Webservice (Work Service)

1. Use MSEWOT to modify work order
    a. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.
    b. Modify with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.

7.2.6. Modify Work Order User Story – Webservice (WorkOrder Service)

2. Use MSEWOT to modify work order
    c. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.
    d. Modify with a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked
        iii. Validate the default flags on Change Account Dialog are set
        iv. Validate +WKR user is checked and transaction rejected
        v. Validate derived Account Code is correct.

7.2.7. Work Order Closure connector (EMM) uses WorkOrder Service

1. Use Work Order Closure Connector (EMM) to modify work order
    e. Modify without a project number
        i. Validate Workgroup and Equipment are mandatory,
        ii. Validate costs are checked

iii. Validate the default flags on Change Account Dialog are set
iv. Validate +WKR user is checked and transaction rejected
v. Validate derived Account Code is correct.

f. Modify with a project number
i. Validate Workgroup and Equipment are mandatory,
ii. Validate costs are checked
iii. Validate the default flags on Change Account Dialog are set
iv. Validate +WKR user is checked and transaction rejected
v. Validate derived Account Code is correct.

7.2.8. Create Work Order (Multi) User Story – Smart Excel (Work Service)

Apply 7.2.1 using 5 or more records

7.2.9. Create Work Order (Multi) User Story – Smart Excel (WorkOrder Service)

Apply 7.2.1 using 5 or more records

7.2.10. Modify Work Order (Multi) User Story – Smart Excel (Work Service)

Apply 7.2.2 using 5 or more records

7.2.11. Modify Work Order (Multi) User Story – Smart Excel (WorkOrder Service)

Apply 7.2.2 using 5 or more records

8. APPENDIX A – CURRENT CODE BASE

8.1. WorkOrderService_create.groovy Hook

```
/**
 * @author Ventyx 2014
 *
 * Pre-Create:
 *     Apply the costing solution for Work Orders to obtain a default Account Code
 *
 *     Note: This will only work if a Work Order is created from scratch. This will
 *     not work when creating a WO using the COPY feature. If this functionality
 *     is required, it will need to be added as an onPostExecute hook to test/modify
 *     the Work Order before the result is returned to the caller.
 *
 *     Modified 10/07/2014 - removed exclusion for Standard Jobs as per HWC request
 *     Modified 23/07/2014 - applied defaults from Standard Job to empty fields
 */

import com.mincom.ellipse.hook.hooks.ServiceHook
import com.mincom.enterpriseservice.ellipse.standardjob.StandardJobServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.workorder.WorkOrderServiceCreateRequestDTO
import com.mincom.enterpriseservice.ellipse.project.ProjectServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.dependant.dto.WorkOrderDTO;
import com.mincom.enterpriseservice.ellipse.equipment.EquipmentServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.workgroup.WorkGroupServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.ErrorMessageDTO
import com.mincom.enterpriseservice.exception.*

public class WorkOrderService_create extends ServiceHook {

/*
 * IMPORTANT!
 * Update this Version number EVERY push to GIT
 */

 private String version = "3"

 private static final Integer ORG_UNIT_LEN = 3
 private static final Integer PROD_OP_ACT_LEN = 7

 @Override
 public Object onPreExecute(Object dto) {
     log.info("WorkOrderService_create onPreExecute - version: ${version}")

     WorkOrderServiceCreateRequestDTO request = dto

           String copyWorkOrder = request.getCopyWorkOrder()?request.getCopyWorkOrder().toString().trim():""
     String districtCode = request.getDistrictCode()?request.getDistrictCode().trim():""
           String standardJob = request.getStdJobNo()?request.getStdJobNo().trim():""
           String projectNo = request.getProjectNo()?request.getProjectNo().trim():""
           String accountCode = request.getAccountCode()?request.getAccountCode().trim():""
           String workGroup = request.getWorkGroup()?request.getWorkGroup().trim():""
     String equipmentRef = request.getEquipmentRef()?request.getEquipmentRef().trim():""
     String equipmentNo = request.getEquipmentNo()?request.getEquipmentNo().trim():""
     // Use Equipment Number if Equipment Reference is not supplied
     if (!equipmentRef) {
         equipmentRef = equipmentNo
     }

           boolean isStandardJob = false
           boolean isProject = false
           boolean isCapital = false
           boolean isEquipment = false
           boolean isWorkGroup = false

           String sjEquipmentRef = ""
           String sjWorkGroup = ""
           String sjProjectNo = ""
           String sjAccountCode = ""

           String projectAccount = ""
           String equipmentAccount = ""
           String workGroupAccount = ""
           String costAccount = ""

     // If copying a Work Order, then have to bypass the rest of the processing
     if (copyWorkOrder) {
         log.info("Copying work order ${copyWorkOrder}")
                   return null
     }

     // If Standard Job entered, get default values from Standard Job
     if (standardJob) {
         log.info("Standard Job entered")
                    (isStandardJob, sjEquipmentRef, sjWorkGroup, sjProjectNo, sjAccountCode) =
getStandardJobDetails(districtCode, standardJob)
                   if (!isStandardJob) {
```

```
                                //Standard Job is not valid - send to Ellipse for standard error
            return null
                    }
                    //Fill empty fields using defaults from Standard Job
                    equipmentRef = equipmentRef?equipmentRef:sjEquipmentRef
                    workGroup = workGroup?workGroup:sjWorkGroup
                    projectNo = projectNo?projectNo:sjProjectNo
                    accountCode = accountCode?accountCode:sjAccountCode
                    log.info("Standard Job defaults applied to empty fields")
    }

    // Equipment Reference is mandatory
    if (equipmentRef.isEmpty()) {
        log.error("Equipment Reference is mandatory")
        //throw EnterpriseServiceOperationException to return to caller
        throw new EnterpriseServiceOperationException(
            new ErrorMessageDTO("", "Equipment Reference is mandatory", "equipmentRef", 0, 0))
    }

    // Work Group is mandatory
    if (workGroup.isEmpty()) {
        log.error("Work Group is mandatory")
        //throw EnterpriseServiceOperationException to return to caller
        throw new EnterpriseServiceOperationException(
            new ErrorMessageDTO("", "Work Group is mandatory", "workGroup", 0, 0))
    }

    // If Project Number entered, get Account Code from Project
    if (projectNo) {
        log.info("Project Number entered")
                    (isProject, isCapital, projectAccount) = getProjectDetails(districtCode, projectNo)
                    if (!isProject) {
                                //Project Number is not valid - send to Ellipse for standard error
            return null
                    }
                    //Set Account Code to Project Account if Capital, or empty for standard costing if not
                    if (isCapital) {
                                if (projectAccount.isEmpty()) {
                                            log.error("No Account Code for Capital Project")
                                            //throw EnterpriseServiceOperationException to return to caller
                                            throw new EnterpriseServiceOperationException(
                                                        new ErrorMessageDTO("", "No Account Code for Capital Project", "projectNo", 0,
0))
                                }
                    }
                    request.setAccountCode(projectAccount)
                    log.info("Work Order Account Code set to Project Account ${projectAccount}")
        return null
    }

    // Check if Account Code entered
    if (accountCode) {
        log.info("Account Code entered ${accountCode}")
                    //Accept entered Account Code
        return null
    }

          (isEquipment, equipmentAccount) = getEquipmentDetails(equipmentRef)
          if (!isEquipment) {
                    //Equipment Reference is not valid - send to Ellipse for standard error
        return null
          }
          if (equipmentAccount.length() < ORG_UNIT_LEN + PROD_OP_ACT_LEN) {
                    log.error("Equipment Account invalid for deriving WO Account")
        //throw EnterpriseServiceOperationException to return to caller
        throw new EnterpriseServiceOperationException(
            new ErrorMessageDTO("", "Equipment Account invalid for deriving WO Account", "equipmentRef", 0, 0))
          }

          (isWorkGroup, workGroupAccount) = getWorkGroupDetails(workGroup)
          if (!isWorkGroup) {
    //Work Group is not valid - send to Ellipse for standard error
        return null
          }
          if (workGroupAccount.length() < ORG_UNIT_LEN) {
                    log.error("Work Group Account invalid for deriving WO Account")
                    //throw EnterpriseServiceOperationException to return to caller
                    throw new EnterpriseServiceOperationException(
                                new ErrorMessageDTO("", "Work Group Account invalid for deriving WO Account", "workGroup", 0, 0))
          }

          //Derive Account Code - chars 1 to 3 of Work Group Account + chars 4 to 10 of Equipment account
          costAccount = workGroupAccount.substring(0, ORG_UNIT_LEN) + equipmentAccount.substring(ORG_UNIT_LEN, ORG_UNIT_LEN +
PROD_OP_ACT_LEN)
          //Set Account Code to derived value
          request.setAccountCode(costAccount)
          log.info("Account Code set to derived value ${costAccount}")
    return null
 }

  /**
  * Read the Standard Job if sent, to check for default fields
  * @param District Code, Standard Job
  * @return Project Number, Account Code, Equipment Reference, Work Group
  */
 private def getStandardJobDetails (String districtCode, String standardJob) {
     log.info("getStandardJobDetails ${districtCode}${standardJob}")
```

```
          boolean isStandardJob = false
          String workGroup = ""
          String projectNo = ""
          String accountCode = ""
          String equipmentRef = ""
          String equipmentNo = ""

    try {
                    StandardJobServiceReadReplyDTO standardJobReply = tools.service.get("StandardJob").read({
                              it.districtCode = districtCode
                              it.standardJob = standardJob})

                    workGroup = standardJobReply.getWorkGroup()?standardJobReply.getWorkGroup().trim():""
                    projectNo = standardJobReply.getProjectNo()?standardJobReply.getProjectNo().trim():""
                    accountCode = standardJobReply.getAccountCode()?standardJobReply.getAccountCode().trim():""
                    equipmentRef = standardJobReply.getEquipmentRef()?standardJobReply.getEquipmentRef().trim():""
               equipmentNo = standardJobReply.getEquipmentNo()?standardJobReply.getEquipmentNo().trim():""
               // Use Equipment Number if Equipment Reference is not supplied
                    equipmentRef = equipmentRef?equipmentRef:equipmentNo
                    isStandardJob = true

    } catch (EnterpriseServiceOperationException e) {
                   log.error("Standard Job not valid")
    }
          return [isStandardJob, equipmentRef, workGroup, projectNo, accountCode]
 }

/**
  * Read the Project and return details
  * @param District Code, Project Number
  * @return Project Exists, Capital Switch, Account Code
  */
 private def getProjectDetails (String districtCode, String projectNo) {
    log.info("getProject ${projectNo}")

          boolean isProject = false
          boolean isCapital = false
          String accountCode = ""
    try {
                    ProjectServiceReadReplyDTO projectReply = tools.service.get("Project").read({
                              it.districtCode = districtCode
            it.projectNo = projectNo})
                    isCapital = projectReply.getCapitalSw()
                    if(isCapital) {
                              accountCode = projectReply.getAccountCode()?projectReply.getAccountCode().trim():""
                    }
                    isProject = true
    } catch (EnterpriseServiceOperationException e) {
                    log.info("Project Number not valid")
    }
          return [isProject, isCapital, accountCode]
 }

 /**
  * Read the Equipment and return details
  * @param Equipment Reference
  * @return Equipment Exists, Account Code
  */
 private def getEquipmentDetails (String equipmentRef) {
    log.info("getEquipmentAccount ${equipmentRef}")

          boolean isEquipment = false
          String accountCode = ""
    try {
                    EquipmentServiceReadReplyDTO equipmentReply = tools.service.get("Equipment").read({
              it.equipmentRef = equipmentRef})
                    accountCode = equipmentReply.getAccountCode()?equipmentReply.getAccountCode().trim():""
                    isEquipment = true
    } catch (EnterpriseServiceOperationException e) {
                    log.info("Equipment Reference not valid")
    }
          return [isEquipment, accountCode]
 }

 /**
  * Read the Work Group and return details
  * @param Work Group
  * @return Work Group Exists, Account Code
  */
 private def getWorkGroupDetails (String workGroup) {
    log.info("getWorkGroupAccount ${workGroup}")

          boolean isWorkGroup = false
          String accountCode = ""
    try {
                    WorkGroupServiceReadReplyDTO workGroupReply = tools.service.get("WorkGroup").read({
              it.workGroup = workGroup})
                    accountCode = workGroupReply.getAccountCode()?workGroupReply.getAccountCode().trim():""
                    isWorkGroup = true
    } catch (EnterpriseServiceOperationException e) {
                    log.info("Work Group not valid")
    }
          return [isWorkGroup, accountCode]
 }

}
```

8.2. WorkOrderService_modify.groovy Hook

```
/**
 * @author Ventyx 2014
 *
 * Pre-Modify:
 *     Apply the costing solution for Work Orders to obtain a default Account Code
 */

import com.mincom.ellipse.hook.hooks.ServiceHook
import com.mincom.enterpriseservice.ellipse.dependant.dto.WorkOrderDTO
import com.mincom.enterpriseservice.ellipse.workorder.WorkOrderServiceModifyRequestDTO
import com.mincom.enterpriseservice.ellipse.workorder.WorkOrderServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.context.ContextServiceFetchContextReplyDTO
import com.mincom.enterpriseservice.ellipse.table.TableServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.project.ProjectServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.equipment.EquipmentServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.workgroup.WorkGroupServiceReadReplyDTO
import com.mincom.enterpriseservice.ellipse.ErrorMessageDTO
import com.mincom.enterpriseservice.exception.*

public class WorkOrderService_modify extends ServiceHook {

/*
 * IMPORTANT!
 * Update this Version number EVERY push to GIT
 */
 private String version = "3"

 private static final String MSF010_TABLE_TYPE_XX = "XX"
 private static final String MSF010_TABLE_TYPE_WKR = "+WKR"
 private static final Integer ORG_UNIT_LEN = 3
 private static final Integer PROD_OP_ACT_LEN = 7

 @Override
 public Object onPreExecute(Object dto) {
     log.info("WorkOrderService_modify onPreExecute - version: ${version}")

     WorkOrderServiceModifyRequestDTO request = dto

     String districtCode = request.getDistrictCode()?request.getDistrictCode().trim():""
     WorkOrderDTO workOrder = request.getWorkOrder()

           boolean useEquipmentNo = false

           String projectNo = request.getProjectNo()?.trim()
           String accountCode = request.getAccountCode()?.trim()
           String workGroup = request.getWorkGroup()?.trim()
     String equipmentRef = request.getEquipmentRef()?.trim()
     String equipmentNo = request.getEquipmentNo()?.trim()
           // Use Equipment Number if Equipment Reference is not supplied
           if (equipmentRef == null) {
                     equipmentRef = equipmentNo
                     useEquipmentNo = true
          }

           String origEquipmentRef = ""
           String origWorkGroup = ""
           String origProjectNo = ""
           String origAccountCode = ""
           BigDecimal actualTotalCost = 0

           boolean isContext = false
           boolean isAuthorised = true
           boolean isWorkOrder = false
           boolean isProject = false
           boolean isCapital = false
           boolean isEquipment = false
           boolean isWorkGroup = false

           String employeeId = ""
           String projectAccount = ""
           String equipmentAccount = ""
           String workGroupAccount = ""
           String costAccount = ""

           log.info("Work Order ${districtCode}${workOrder.toString()}")

     // Get existing Work Order costing details
           (isWorkOrder, origEquipmentRef, origWorkGroup, origProjectNo, origAccountCode, actualTotalCost) =
getWorkOrderDetails(districtCode, workOrder, useEquipmentNo)
           if (!isWorkOrder) {
                     //Work Order is not valid - send to Ellipse for standard error
                     return null
           }

     // Default any attributes not sent in request
           if (projectNo == null) {projectNo = origProjectNo}
           if (accountCode == null) {accountCode = origAccountCode}
           if (workGroup == null) {workGroup = origWorkGroup}
           if (equipmentRef == null) {equipmentRef = origEquipmentRef}

     // Equipment Reference is mandatory
     if (equipmentRef.isEmpty()) {
```

```
                    log.error("Equipment Reference is mandatory")
          //throw EnterpriseServiceOperationException to return to caller
          throw new EnterpriseServiceOperationException(
              new ErrorMessageDTO("", "Equipment Reference is mandatory", "equipmentRef", 0, 0))
     }

     // Work Group is mandatory
     if (workGroup.isEmpty()) {
                    log.error("Work Group is mandatory")
                    //throw EnterpriseServiceOperationException to return to caller
                    throw new EnterpriseServiceOperationException(
                              new ErrorMessageDTO("", "Work Group is mandatory", "workGroup", 0, 0))
     }

     // Check if Work Order costing details will change
           if (equipmentRef.equals(origEquipmentRef) &&
                    workGroup.equals(origWorkGroup) &&
                    projectNo.equals(origProjectNo) &&
                    accountCode.equals(origAccountCode)) {
                    log.info("No costing details changed")
                    return null
           }

           // Check if Work Order has existing costs
           if (actualTotalCost > 0) {
                    // Determine Employee Id for the logged-in user
                    (isContext, employeeId) = getContextDetails(districtCode)
                    // Check employee authority level
                    if (isContext) {
                              isAuthorised = getEmployeeAuthority(employeeId)
                    } else {
                              log.info("Cannot determine employee, default to no authority")
                              isAuthorised = false
                    }
           }

     // If Project Number modified, get Account Code from Project
     if (!projectNo.equals(origProjectNo)) {
                    //Reject if user not authorised to modify costing
                    if (!isAuthorised) {
                              log.error("Costs exist, not authorised to modify Project Number")
                              //throw EnterpriseServiceOperationException to return to caller
                              throw new EnterpriseServiceOperationException(
                                        new ErrorMessageDTO("", "Costs exist, not authorised to modify Project Number",
"projectNo", 0, 0))
                    }
                if (projectNo) {
                              (isProject, isCapital, projectAccount) = getProjectDetails(districtCode, projectNo)
                              if (!isProject) {
                                        //Project Number is not valid - send to Ellipse for standard error
                      return null
                              }
                              //Set Account Code to Project Account if Capital, or empty for standard costing if not
                              if (isCapital) {
                                        if (projectAccount.isEmpty()) {
                                                  log.error("No Account Code for Capital Project")
                                                  //throw EnterpriseServiceOperationException to return to caller
                                                  throw new EnterpriseServiceOperationException(
                                                            new ErrorMessageDTO("", "No Account Code for Capital Project",
"projectNo", 0, 0))
                                        }
                              }
                              request.setAccountCode(projectAccount)
                              log.info("Work Order Account Code set to Project Account ${projectAccount}")
                              return null
                }
     }

     // Check if Account Code changed
     if (!accountCode.equals(origAccountCode)) {
                    //Reject if user not authorised to modify costing
                    if (!isAuthorised) {
                              log.error("Costs exist, not authorised to modify Account Code")
                              //throw EnterpriseServiceOperationException to return to caller
                              throw new EnterpriseServiceOperationException(
                                        new ErrorMessageDTO("", "Costs exist, not authorised to modify Account Code",
"accountCode", 0, 0))
                    }
                    // Check if Project Costing in use
                    if (projectNo) {
                              log.error("Cannot change Account Code, Project costing in use")
                              //throw EnterpriseServiceOperationException to return to caller
                              throw new EnterpriseServiceOperationException(
                                        new ErrorMessageDTO("", "Cannot change Account Code, Project costing in use",
"accountCode", 0, 0))
                    }
                if (accountCode) {
                    log.info("Account Code entered ${accountCode}")
                              //Accept entered Account Code
                    return null
                }
           }

           // Check if Project Costing in use
           if (projectNo) {
          log.info("Project costing in use, no effect from Equip/WrkGrp change")
                    //Accept changes to Equipment/Work Group, no impact on costing
```

```
          return null
                    }

            (isEquipment, equipmentAccount) = getEquipmentDetails(equipmentRef)
            if (!isEquipment) {
                    //Equipment Reference is not valid - send to Ellipse for standard error
          return null
            }
            if (equipmentAccount.length() < ORG_UNIT_LEN + PROD_OP_ACT_LEN) {
                    log.error("Equipment Account invalid for deriving WO Account")
          //throw EnterpriseServiceOperationException to return to caller
          throw new EnterpriseServiceOperationException(
              new ErrorMessageDTO("", "Equipment Account invalid for deriving WO Account", "equipmentRef", 0, 0))
            }

            (isWorkGroup, workGroupAccount) = getWorkGroupDetails(workGroup)
            if (!isWorkGroup) {
                    //Work Group is not valid - send to Ellipse for standard error
          return null
            }
            if (workGroupAccount.length() < ORG_UNIT_LEN) {
                    log.error("Work Group Account invalid for deriving WO Account")
                    //throw EnterpriseServiceOperationException to return to caller
                    throw new EnterpriseServiceOperationException(
                            new ErrorMessageDTO("", "Work Group Account invalid for deriving WO Account", "workGroup", 0, 0))
            }

            //Derive Account Code - chars 1 to 3 of Work Group Account + chars 4 to 10 of Equipment account
            costAccount = workGroupAccount.substring(0, ORG_UNIT_LEN) + equipmentAccount.substring(ORG_UNIT_LEN, ORG_UNIT_LEN +
PROD_OP_ACT_LEN)
            if (!accountCode) {
                    //Set Account Code to derived value and send to Ellipse
                    request.setAccountCode(costAccount)
                    log.info("Account Code set to derived value ${costAccount}")
                return null
            }
            if (!costAccount.equals(accountCode)) {
                    if (!isAuthorised) {
                            log.error("Costs exist, not authorised to modify Equip/WrkGrp")
                            //throw EnterpriseServiceOperationException to return to caller
                            throw new EnterpriseServiceOperationException(
                                    new ErrorMessageDTO("", "Costs exist, not authorised to modify Equip/WrkGrp",
"equipmentRef", 0, 0))
                    }
            }
            //Accept changes to Equipment Reference and/or Work Group
            return null
 }

 /**
  * Read the Work Order to check for modified fields
  * @param District Code, Work Order
  * @return Project Number, Account Code, Equipment Reference, Work Group
  */
 private def getWorkOrderDetails (String districtCode, WorkOrderDTO workOrder, boolean useEquipmentNo) {
     log.info("getWorkOrderDetails ${districtCode}${workOrder}")

            boolean isWorkOrder = false
            String equipmentRef = ""
            String workGroup = ""
            String projectNo = ""
            String accountCode = ""
            BigDecimal actualTotalCost = 0

     try {
                    WorkOrderServiceReadReplyDTO workOrderReply = tools.service.get("WorkOrder").read({
                            it.districtCode = districtCode
                            it.workOrder = workOrder})

                    if (useEquipmentNo) {
                            equipmentRef = workOrderReply.getEquipmentNo()?workOrderReply.getEquipmentNo().trim():""
                    } else {
                            equipmentRef = workOrderReply.getEquipmentRef()?workOrderReply.getEquipmentRef().trim():""
                    }
                    workGroup = workOrderReply.getWorkGroup()?workOrderReply.getWorkGroup().trim():""
                    projectNo = workOrderReply.getProjectNo()?workOrderReply.getProjectNo().trim():""
                    accountCode = workOrderReply.getAccountCode()?workOrderReply.getAccountCode().trim():""
                    actualTotalCost = workOrderReply.actualEquipmentCost + workOrderReply.actualLabCost +
workOrderReply.actualMatCost + workOrderReply.actualOtherCost
                    isWorkOrder = true

     } catch (EnterpriseServiceOperationException e) {
                   log.error("Work Order not valid")
     }
            return [isWorkOrder, equipmentRef, workGroup, projectNo, accountCode, actualTotalCost]
 }

 /**
  * Fetch Context data to get Employee Id for logged-in user
  * @param District Code
  * @return Context Found, Employee Id
  */
 private def getContextDetails (String districtCode) {
     log.info("getContextDetails for logged-in user")

            boolean isContext = false
            String employeeId = ""
```

```
    try {
                    ContextServiceFetchContextReplyDTO contextReply = tools.service.get("Context").fetchContext({
                            it.district = districtCode})
                    employeeId = contextReply.getEmployeeId()
                    isContext = true
    } catch (EnterpriseServiceOperationException e) {
                   log.info("Cannot get Context to determine Employee Id")
    }
          return [isContext, employeeId]
}

/**
 * Read the +WKR Table to check if Employee is authorised
 * @param Employee Id
 * @return Employee Authorised
 */
private def getEmployeeAuthority (String employeeId) {
    log.info("getEmployeeAuthority ${employeeId}")

          boolean isAuthorised = true

    try {
                    TableServiceReadReplyDTO tableReply = tools.service.get("Table").read({
                            it.tableType = MSF010_TABLE_TYPE_XX
                            it.tableCode = MSF010_TABLE_TYPE_WKR})
                    isAuthorised = false
                    TableServiceReadReplyDTO codeReply = tools.service.get("Table").read({
                            it.tableType = MSF010_TABLE_TYPE_WKR
                            it.tableCode = employeeId})
                    isAuthorised = true
    } catch (EnterpriseServiceOperationException e) {
                   log.error("Employee not authorised to change costing")
    }
          return isAuthorised
}

/**
 * Read the Project and return details
 * @param District Code, Project Number
 * @return Project Exists, Capital Switch, Account Code
 */
private def getProjectDetails (String districtCode, String projectNo) {
    log.info("getProjectDetails ${projectNo}")

          boolean isProject = false
          boolean isCapital = false
          String accountCode = ""

    try {
                    ProjectServiceReadReplyDTO projectReply = tools.service.get("Project").read({
            it.districtCode = districtCode
            it.projectNo = projectNo})
                    isCapital = projectReply.getCapitalSw()
                    if(isCapital) {
                            accountCode = projectReply.getAccountCode()?projectReply.getAccountCode().trim():""
                    }
                    isProject = true
    } catch (EnterpriseServiceOperationException e) {
                   log.error("Project Number not valid")
    }
          return [isProject, isCapital, accountCode]
}

/**
 * Read the Equipment and return details
 * @param Equipment Reference
 * @return Equipment Exists, Account Code
 */
private def getEquipmentDetails (String equipmentRef) {
    log.info("getEquipmentDetails ${equipmentRef}")

          boolean isEquipment = false
          String accountCode = ""

    try {
                    EquipmentServiceReadReplyDTO equipmentReply = tools.service.get("Equipment").read({
            it.equipmentRef = equipmentRef})
                    accountCode = equipmentReply.getAccountCode()?equipmentReply.getAccountCode().trim():""
                    isEquipment = true
    } catch (EnterpriseServiceOperationException e) {
                   log.error("Equipment Reference not valid")
    }
          return [isEquipment, accountCode]
}

/**
 * Read the Work Group and return details
 * @param Work Group
 * @return Is Work Group, Account Code
 */
private def getWorkGroupDetails (String workGroup) {
    log.info("getWorkGroupDetails ${workGroup}")

          boolean isWorkGroup = false
          String accountCode = ""
```

```
    try {
                    WorkGroupServiceReadReplyDTO workGroupReply = tools.service.get("WorkGroup").read({
            it.workGroup = workGroup})
                    accountCode = workGroupReply.getAccountCode()?workGroupReply.getAccountCode().trim():""
                    isWorkGroup = true
    } catch (EnterpriseServiceOperationException e) {
                   log.error("Work Group not valid")
    }
          return [isWorkGroup, accountCode]
 }
}
```

8.3. WorkService_create.groovy

```
/**
 **  WorkService_create.groovy
 **
 **  DESCRIPTION:
 **  ------------
 **  Applies mandatory validation for Job Code 10 (Safety Hazard).
 **
 **  REVISION HISTORY:
 **  -----------------
 **  23-Nov-2021   J Whitting        Version 3 - skip mandatory check for Safety Hazard
 **                                  for Save As (i.e. Copy Work Order) functionality
 **  29-Sep-2021   J Whitting        Version 2 - skip mandatory check for Safety Hazard
 **                                  if the value can be sourced from the Standard Job
 **  13-Apr-2021   P.Cranna          Version 1.
 **  ............................... Initial Coding.
 **
 **/

import com.mincom.ellipse.hook.hooks.ServiceHook
import com.mincom.enterpriseservice.exception.*
import com.mincom.ellipse.types.m3620.instances.WorkServiceResult
import com.mincom.ellipse.types.m3620.instances.WorkDTO
import com.mincom.enterpriseservice.ellipse.standardjob.StandardJobServiceReadReplyDTO
import com.mincom.ellipse.errors.exceptions.FatalException
import com.mincom.enterpriseservice.ellipse.ErrorMessageDTO
import com.mincom.enterpriseservice.ellipse.InformationMessageDTO
import com.mincom.enterpriseservice.exception.*
import com.mincom.ellipse.errors.Message
import com.mincom.ellipse.errors.Error

class WorkService_create extends ServiceHook {

    private String hookVersion = "3"

    @Override
    public Object onPreExecute(Object dto) {
        log.info("Hooks onPreExecute - version: ${hookVersion}")

        WorkDTO request = (WorkDTO) dto

        String copyWorkOrder = request.getCopyWorkOrder().getValue()
        String jobCode10 = request.getJobCode10().getValue()
        String districtCode = request.getDistrictCode().getValue()
        String stdJobNo = request.getStdJobNo().getValue()

        // If copying a Work Order, then bypass the validation
        if (copyWorkOrder?.trim()) {
            return null
        }

        // Check whether Safety Hazard can be sourced from the Standard Job
        if (!jobCode10?.trim() && stdJobNo?.trim()) {
            try {
                StandardJobServiceReadReplyDTO stdJobReply = tools.service.get('StandardJob').read({
                    it.districtCode = districtCode
                    it.standardJob = stdJobNo
                })
                jobCode10 = stdJobReply.getJobCode10()
            } catch(Exception e) {
                jobCode10 = ""
            }
        }

        if (!jobCode10?.trim()) {
            log.info("Error - Safety Hazard [jobCode10] is mandatory")
            // throw new FatalException("INPUT REQUIRED - SAFETY HAZARD (WORK ORDER JOB CODE 10)]")
            // The FatalException will return a clean error message to the UI, but
            // won't position the cursor in the error field
```

```
            // Build the reply with errors
            WorkServiceResult reply = new WorkServiceResult()
            reply.setWorkDTO(request)

            // Return error
            com.mincom.ellipse.errors.Error[] arrayOfErrors = new com.mincom.ellipse.errors.Error[1]
            List errParameters = ["INPUT REQUIRED - SAFETY HAZARD (WORK ORDER JOB CODE 10)"]
            arrayOfErrors[0] = new com.mincom.ellipse.errors.Error("M3620.E0038", errParameters)
            arrayOfErrors[0].fieldId = "jobCode10"
            arrayOfErrors[0].fieldIndex = "0"
            reply.setErrors(arrayOfErrors)

            return reply
        }

        return null
    }


}
```

8.4. WorkService_update.groovy (Note: Not in PROD)

```
/**
 **  WorkService_update.groovy
 **
 **  DESCRIPTION:
 **  ------------
 **  Applies mandatory validation for Job Code 10 (Safety Hazard).
 **
 **  REVISION HISTORY:
 **  -----------------
 **  13-Apr-2021   P.Cranna          Version 1.
 **  ............................... Initial Coding.
 **
 **/

import com.mincom.ellipse.hook.hooks.ServiceHook
import com.mincom.enterpriseservice.exception.*
import com.mincom.ellipse.types.m3620.instances.WorkServiceResult
import com.mincom.ellipse.types.m3620.instances.WorkDTO
import com.mincom.ellipse.errors.exceptions.FatalException
import com.mincom.enterpriseservice.ellipse.ErrorMessageDTO
import com.mincom.enterpriseservice.ellipse.InformationMessageDTO
import com.mincom.enterpriseservice.exception.*
import com.mincom.ellipse.errors.Message
import com.mincom.ellipse.errors.Error

class WorkService_update extends ServiceHook {

    private String hookVersion = "1c"

    @Override
    public Object onPreExecute(Object dto) {
        log.info("Hooks onPreExecute - version: ${hookVersion}")

        WorkDTO request = (WorkDTO) dto

        String jobCode10 = request.getJobCode10().getValue()

        if (!jobCode10?.trim()) {
            log.info("Error - Safety Hazard [jobCode10] is mandatory")
            // throw new FatalException("INPUT REQUIRED - SAFETY HAZARD (WORK ORDER JOB CODE 10)")
            // The FatalException will return a clean error message to the UI, but
            // won't position the cursor in the error field

            // Build the reply with errors
            WorkServiceResult reply = new WorkServiceResult()
            reply.setWorkDTO(request)

            // Return error
            com.mincom.ellipse.errors.Error[] arrayOfErrors = new com.mincom.ellipse.errors.Error[1]
            List errParameters = ["INPUT REQUIRED - SAFETY HAZARD (WORK ORDER JOB CODE 10)"]
            arrayOfErrors[0] = new com.mincom.ellipse.errors.Error("M3620.E0038", errParameters)
            arrayOfErrors[0].fieldId = "jobCode10"
            arrayOfErrors[0].fieldIndex = "0"
            reply.setErrors(arrayOfErrors)

            return reply
        }

        return null
    }


}
```

Hunter Water
ABN 46 228513 446
Customer enquiries 1300 657 657
enquiries@hunterwater.com.au
hunterwater.com.au

Version: 2

Date: FEBRUARY 2022