

On Key 5 Web Service API Guide

Document Information	
Title:	On Key 5 Web Service API Guide
Revision date:	2021/01/29
Owner:	Pragma Products

Change History	
2012/10/29	Initial Implementation
2012/11/05	Changed location of List of Services and added server location of different On Key documents referenced
2013/02/19	Added Change Log
2013/08/20	Added Release 5.8 Change Log entries
2013/10/21	Added Release 5.8 SP1 Change Log entries
2014/02/13	Added Release 5.9 Change Log entries
2014/04/29	Added Release 5.9 SP1 Change Log entries
2014/08/19	Added Release 5.10 Change Log entries
2014/10/28	Added Release 5.10 SP1 Change Log entries
2015/03/02	Added Release 5.11 Change Log entries
2015/04/30	Added Release 5.11 SP1 Change Log entries
2015/08/24	Added Release 5.12 Change Log entries
2015/11/02	Added Release 5.12 SP1 Change Log entries
2016/03/03	Added Release 5.13 Change Log entries
2016/04/18	Added Release 5.13 SP1 Change Log entries
2017/06/23	Added Releases 5.13 SP2, 5.13 SP3, 5.13 SP4, 5.13 SP5, 5.14 SP0, 5.14 SP1 and some 5.15 SP0 Change Log entries
2018/04/17	Added Release 5.16 SP1 and 5.16 SP2 change log entries
2018/07/16	Added Release 5.16 SP2 change log entries
2021/01/29	Added Release 5.18 SP3 change log entries

Contents

1. Introducing On Key Web Service API.....	3
1.1 Conventions Used in This Guide.....	3
1.2 When to Use the Web Service API	3
1.3 Standards Compliance	3
1.4 Support Policy	4
1.5 Quick Start.....	4
1.5.1 Install On Key	4
1.5.2 Generate and Import WSDL into Your Development Platform	4
1.6 Walk Through Sample Code	4
2. On Key Web Service API Style	9
2.1 Synchronous (Request/Response)	9
2.2 Asynchronous (Request/Acknowledge/Poll)	9
3. Security.....	10
3.1 Walk Through Authentication	10
4. Error Handling.....	11
4.1 Session Expiration.....	11
5. Reference	12
5.1 Imports.....	12
5.2 Exports	12
5.2.1 Export Data Request Message	12
5.2.2 Export Data Response Message.....	12
5.3 Fault Codes	13
5.4 List of Services	14
6. Change Log	15

1. Introducing On Key Web Service API

Welcome to the On Key Web Service API guide. The purpose behind this guide is to provide a succinct introduction to building solutions that integrate with the Web Services published as part of an On Key deployment.

1.1 Conventions Used in This Guide

The following typographical conventions are used in this guide:

Code - Used to indicate C# code samples, command line statements, and other language keywords

The following keywords are used in this guide:

<vdir> - The IIS virtual directory that hosts the On Key server (e.g. <https://MyServer.Domain/OnKey5>)

<ok_guide> - The On Key Installation Guide located at **<vdir>/Docs/OnKey_InstallationGuide.pdf**

<ok_manual> - The On Key User Manual located at **<vdir>/Help/index.htm**

<it_manual> - The Interface Tool User Manual located at **<vdir>/Docs/InterfaceTool_UserManual.pdf**

<sys_manual> - The SYSPRO User Manual located at **<vdir>/Docs/SYSPRO_UserManual.pdf**

1.2 When to Use the Web Service API

The On Key Web services provide the ability to import/export data from a wide variety of Business Objects within On Key. These Data Integration Web Services are ideally suited for programmatically interfacing between On Key and external systems like an ERP. Besides the programmatic access provided by these Web services, On Key also provides a standalone application called the Interface Tool that can be used to import/export data within On Key.

The Interface Tool reads and writes Tab Delimited UTF 8 files and calls the same set of On Key Data Integration Web Services to allow interfacing with external systems. The Interface Tool can be configured and scheduled to run via Windows Scheduler and provides great support for error handling, file archiving, viewing log files and much more. The same set of functionality can also be executed via a simple, intuitive user interface. The Interface Tool also supports a SYSPRO plugin that provides direct integration with the [SYSPRO ERP](#) system via their SYSPRO Web Service API. More information on the Interface Tool and SYSPRO plugin can be found in the **<it_manual>** and **<sys_manual>** documents.

Clients can therefore either use the On Key Data Integration Web Services directly through the Web Service API or alternatively use the file based integration supported by the Interface Tool. As the Interface Tool is a standalone application, it will be more difficult to fit into a SOA architecture where integration happens via Web services. For these environments using the Web Service API makes the most sense.

1.3 Standards Compliance

The On Key Web Services are implemented to be fully interoperable across different technology stacks by complying with the following specifications:

1. Simple Object Access Protocol (SOAP) 1.1
2. Web Service Description Languages (WSDL) 1.1
3. WS-I Basic Profile 1.1

The Web services should therefore work with current SOAP development environments, including, but not

limited to Visual Studio .NET and Java based development environments. All code samples provided in this document are in C# (.NET).

1.4 Support Policy

Pragma does not plan on supporting multiple versions of the Web Service API in On Key. Clients should always use the latest version of the Web Service API published with On Key releases to fully exploit the benefits of richer features and greater efficiency. In evolving the Web Service API, Pragma will strive to ensure backwards compatibility. This flexible strategy will allow compatible changes to be made to the Web Service API without forcing a new contract version. Breaking changes will however result in a new contract version and will be indicated as part of the release documentation.

1.5 Quick Start

1.5.1 Install On Key

Follow the steps detailed in the [ok_guide](#) to install On Key on a server running Internet Information Services. Ensure that the installation is working by making sure that users can successfully log into On Key by browsing to [vdir](#) using your internet browser. Refer to the [ok_guide](#) for information on what internet browsers are supported by On Key.

1.5.2 Generate and Import WSDL into Your Development Platform

To access the On Key Web Service, you need a Web Service Description Language (WSDL) file that defines the service operations available for consumption. This WSDL file is imported by your development platform and used to generate the necessary objects for use in building client Web service applications.

The WSDL for a particular On Key Web service can be generated at:

[vdir](#)/Services/Interfaces/<servicename.svc>?singleWsd1, e.g.
<https://myserver.domain/OnKey5/Services/Interfaces/Authentication.svc?singleWsd1>.

Refer to [List of Services](#) for a complete list of available services.

Using the above information, here is a sample command that illustrates the use of the `wsimport` tool to generate JAX-WS portable artefacts for Java based environments from the command line:

```
wsimport.exe, -p, Authentication, -s, C:\Proxies, -verbose,  
https://myserver.domain/OnKey5/Services/Interfaces/Authentication.svc?singleWsd1
```

For .NET environments, the same can be accomplished on the command line by using the `svcutil.exe` utility (refer to [documentation](#)) or by using the Add Service Reference functionality directly in the Visual Studio IDE.

1.6 Walk Through Sample Code

This following C# program shows a sample client application that demonstrates the invocation and handling of several of the common On Key Web Service API calls. The sample application performs the following tasks:

1. Logs into On Key using the [Authentication Service](#) to establish a valid client session.
2. Exports some data from On Key by using the [Export Service](#) to run an On Key Analyser Report.
3. Ends the client session by logging off from On Key using the [Authentication Service](#).

```
using System;
using System.Collections.ObjectModel;
using System.ServiceModel;
using Pragma.OnKey.Core.Exceptions;
using Pragma.OnKey.Model.Common;
using Pragma.OnKey.Sdk;
using Pragma.OnKey.Services.Common.Authentication;
using Pragma.OnKey.Services.Common.Export;

namespace Pragma.OnKey.Services.ApiSample
{
    public class WebServicesApiSample
    {
        public string SessionId { get; set; }

        public static void Start()
        {
            Console.WriteLine("Enter user name:");
            string userName = Console.ReadLine();

            Console.WriteLine("Enter password:");
            string password = Console.ReadLine();

            Console.WriteLine("Enter connection name:");
            string connectionName = Console.ReadLine();

            WebServicesApiSample sample = new WebServicesApiSample();
            sample.RunApiSamples(userName, password, connectionName);
        }

        public void RunApiSamples(string userName, string password, string connectionName)
        {
            if (!Logon(userName, password, connectionName))
                return;

            ExportDataSet dataSet = ExportWorkOrders(SessionId, "B%");
            if (dataSet != null)
                Console.WriteLine("Data Xml: {0}", dataSet.Data);

            Logoff(SessionId);
        }

        public bool Logon(string userName, string password, string connectionName)
        {
            AuthenticationService proxy = new AuthenticationService();
            try
            {
                var credentials = new LogonDetails();
                credentials.UserName = userName;
                credentials.Password = password;
                credentials.ConnectionName = connectionName;

                var request = new LogonRequest(credentials);
```

```
        LogonResponse response = proxy.Logon(request);

        if (response.Errors.Count > 0)
        {
            HandleErrors(response.Errors);
            return false;
        }

        SessionId = response.SessionId;
        return true;
    }
    catch (FaultException<OnKeyServiceFault> ex)
    {
        HandleServiceException(ex);
        return false;
    }
    finally
    {
        ((ICommunicationObject)proxy).Close();
    }
}

public void Logoff(string sessionId)
{
    AuthenticationService proxy = new AuthenticationService();
    try
    {
        var request = new LogOffRequest(sessionId);
        LogOffResponse response = proxy.LogOff(request);

        if (response.Errors.Count > 0)
        {
            HandleErrors(response.Errors);
        }
    }
    catch (FaultException<OnKeyServiceFault> ex)
    {
        HandleServiceException(ex);
    }
    finally
    {
        ((ICommunicationObject)proxy).Close();
    }
}

public ExportDataSet ExportWorkOrders(string sessionId, string assetCode)
{
    ExportService proxy = new ExportService();
    try
    {
        var request = new ExportDataRequest();

        // Required for authentication
```

```

        request.SessionId = sessionId;

        request.ReportCode = "EXPORT WORK ORDERS";
        request.DataSetName = "WorkOrderAsset";
        request.MaxRecordCount = 200;
        request.Parameters = new Collection<ExportQueryParameter> {
            new ExportQueryParameter { Name = "AssetCode", Value = assetCode }
        };

        ExportDataResponse response = proxy.ExportData(request);

        if (response.Errors.Count == 0)
            return response.DataSet;
        else
        {
            HandleErrors(response.Errors);
            return null;
        }
    }
    catch (FaultException<OnKeyServiceFault> ex)
    {
        HandleServiceException(ex);
        return null;
    }
    finally
    {
        ((ICommunicationObject)proxy).Close();
    }
}

public static void HandleServiceException(FaultException<OnKeyServiceFault> ex)
{
    // All exceptions that occur on the On Key Server are thrown as
    // FaultException<OnKeyServiceFault> exceptions to clients
    // Well-known server side exceptions are indicated through the use
    // of the OnKeyFaultCode. This property may be or may be not set.

    // Determine whether this is a well-known server side exception
    if (!string.IsNullOrEmpty(ex.Detail.ErrorCode))
    {
        string message = ex.Detail.ErrorMessage;
        string faultCode = ex.Detail.ErrorCode;
        switch (faultCode)
        {
            case "SessionExpired":
                Console.WriteLine("User's session has timed out: {0}", message);
                break;
            case "RecordNotFound":
                Console.WriteLine("Cannot find the record requested: {0}",
message);
                break;
            case "TooManyConcurrentSessions":
                Console.WriteLine("Too many sessions for the same user: {0}",

```

```

message);
        break;
    case "DatabaseConcurrencyConflict":
        Console.WriteLine("Another user updated the same record: {0}",
message);
        break;
    case "LogonPasswordExpired":
        Console.WriteLine("User's password has expired: {0}", message);
        break;
    case "NotAuthorized":
        Console.WriteLine("User not authorized: {0}", message);
        break;
    default:
        Console.WriteLine("Server side exception occurred: {0}", message);
        break;
    }
}
else
{
    // All other unexpected server side errors
    string message = ex.Detail != null ? ex.Detail.ErrorMessage : ex.Message;
    Console.WriteLine("Server side exception occurred: {0}", message);
}
}

public static void HandleErrors(Collection<string> errors)
{
    foreach (var error in errors)
    {
        Console.WriteLine("Error: {0}", error);
    }
}
}
}

```

2. On Key Web Service API Style

The On Key Web Service API calls are implemented using Request/Response Messages. The client application submits a Service request using a Request Message; the On Key Web Service processes the Request and returns a Response Message that can be handled by the client application. The Response Message can contain business errors that occurred on the On Key Server. Unexpected exceptions are returned as SOAP faults – refer to [Error Handling](#) for more information.

Most of the Web Services support both [Asynchronous \(Request/Acknowledge/Poll\)](#) and [Synchronous \(Request/Response\)](#) call patterns.

2.1 Synchronous (Request/Response)

When using the Synchronous API, the client application sends a Service request and blocks/waits while waiting for a response. The response is sent over the same client connection and therefore suited for service operations where the average time to return a response is relatively short.

2.2 Asynchronous (Request/Acknowledge/Poll)

When using the Asynchronous API, the client application sends a Service request that is forwarded to a background process for processing on the server. The client application receives an acknowledgement via a Request Identifier (RID). The client application uses the RID to periodically poll the server for progress. Once the operation has completed, the client application uses the RID to retrieve the results.

By separating in time the receipt of the request, the processing of the request and delivery of the response, the asynchronous API is ideally suited for longer running service operations and safeguards the On Key server against spikes in request load.

3. Security

Client applications that make use of the On Key Web Services are subject to the same security measures implemented by On Key. Network traffic flowing over the wire is encrypted by using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.

For User authentication, client applications must log in using the valid credentials for an On Key User account. If successful, the client application is provided with a `SessionId` that must be set into the Session header for all subsequent Web service calls. Authorisation for the different Service operations is governed by the User Rights associated with the authenticated On Key User.

For more information on how to configure a User with appropriate User Rights in On Key, refer to the [<ok_manual>](#). For more information on Session management within On Key, please refer to the *Configure On Key Application Server Settings* section in the [<ok_guide>](#).

3.1 Walk Through Authentication

Consider the following C# sample that illustrates how to use the Authentication service to log into On Key to retrieve a valid On Key `SessionId`.

```
public static string Logon(string userName, string password, string connectionName)
{
    AuthenticationService proxy = new AuthenticationService();
    try
    {
        var credentials = new LogonDetails();
        credentials.UserName = userName;
        credentials.Password = password;
        credentials.ConnectionName = connectionName;

        var request = new LogonRequest(credentials);
        LogonResponse response = proxy.Logon(request);

        if (response.Errors.Count == 0)
            return response.SessionId;
        else
        {
            HandleErrors(response.Errors);
            return null;
        }
    }
    catch (FaultException<OnKeyServiceFault> ex)
    {
        HandleServiceException(ex);
        return null;
    }
    finally
    {
        ((ICommunicationObject)proxy).Close();
    }
}
```

4. Error Handling

The Web Service API calls return error information that a client application can use to identify and resolve run-time errors. The following kinds of error information can be returned:

- For unhandled exceptions that occur on the server, the API returns a SOAP fault message with an associated [OnKeyFaultCode](#).
- For most other errors like missing fields, invalid data formats or any other business rule violations, the API returns a collection of Error records with their associated error messages.

4.1 Session Expiration

When a client application logs into On Key using the [Authentication service](#), a new client session is created on the On Key server. The client session is set to expire after a period of inactivity. The default period is 30 minutes but can be configured as documented in the *Configure On Key Application Server Settings* section in the [<ok_guide>](#).

When a session expires, a SOAP fault with an **ErrorCode** of **SessionExpired** will be returned. If this happens, you must call the [Logon service](#) operation again to establish a new session for the client application.

5. Reference

5.1 Imports

Each Business Object in On Key has its own Import web service that allows a client application to send through a batch of records for processing to the On Key server. All the Import service operations support both [synchronous](#) and [asynchronous](#) call patterns. When a client application triggers an Import, the service operation will perform the action by simulating what the user would have done when doing a similar action via the On Key user interface.

The end result is that all normal functionality in On Key is triggered when executing the service operation. If the service operation violates the On Key business rules, normal error messages will be passed back to the client application. These error messages will be contained in the [RecordFailures](#) collection associated with every Import response message.

In addition to error messages, client applications can also opt to receive success notifications for every imported record by setting the [IncludeRecordSuccesses](#) property associated with every Import request message. The [ImportRecordSuccess](#) records will also contain the unique [RecordId](#) for records that have been inserted into, merged (inserted/updated) or deleted from the On Key database.

Refer to the Import Template files that are included with the Interface Tool for more information on what fields are mandatory for the different Import services.

5.2 Exports

Instead of providing corresponding Export web services for all Business Objects in On Key, all export requests are handled by a single generic Export web service. The client defines what data to export by writing SQL queries using the On Key Analyser Reports functionality provided by the On Key user interface. These queries are executed by invoking the [ExportData](#) service operation. Refer to the [<ok_manual>](#) for more information on writing Analyser Reports.

5.2.1 Export Data Request Message

To execute an export, the client application needs to send through the [ReportCode](#) for the Analyser Report as well as any [Parameters](#) in the [ExportDataRequest](#) message. The [MaxRecordCount](#) property can be used to limit the number of records returned in the [ExportDataResponse](#) message. The [DataSetName](#) can be used to name the XML element containing the [exported data](#) results.

Consider the following C# snippet that illustrates how to populate an [ExportDataRequest](#) message:

```
var request = new ExportDataRequest();
request.ReportCode = "EXPORT WORK ORDERS";
request.DataSetName = "WorkOrderAsset";
request.MaxRecordCount = 200;
request.Parameters = new Collection<ExportQueryParameter> {
    new ExportQueryParameter { Name = "AssetCode", Value = "B%" }
};
```

```
ExportDataResponse response = ExportService.ExportData(request);
```

5.2.2 Export Data Response Message

The [ExportDataResponse](#) will contain the records in a self-describing XML format contained within the generic [DataSet](#) structure. The [Schema](#) property of the [DataSet](#) describes the XML structure for the data returned in the [Data](#) property. Client applications can use XPath queries to extract the relevant information from the

response data received.

Consider the `ExportDataResponse` XML extracted from the SOAP message returned by the [export request](#):

```
<ExportDataResponse xmlns="http://schemas.pragmaproducts.com/onkey/System/v1">
  <DataSet xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <Data><![CDATA[<WorkOrderAssetDataSet>
      <WorkOrderAsset>
        <Code>S00122</Code>
        <AssetCode>B0IL001</AssetCode>
      </WorkOrderAsset>
      <WorkOrderAsset>
        <Code>S00123</Code>
        <AssetCode>B0IL001</AssetCode>
      </WorkOrderAsset>
      <WorkOrderAsset>
        <Code>S00124</Code>
        <AssetCode>B0IL001</AssetCode>
      </WorkOrderAsset>
    ]]></Data>
    <Schema>
      <Columns>
        <Column>
          <ColumnName>Code</ColumnName>
          <DataType>String</DataType>
        </Column>
        <Column>
          <ColumnName>AssetCode</ColumnName>
          <DataType>String</DataType>
        </Column>
      </Columns>
      <DataSetName>WorkOrderAsset</DataSetName>
    </Schema>
  </DataSet>
  <Errors xmlns="http://schemas.pragmaproducts.com/onkey/v1"
    xmlns:a="http://schemas.microsoft.com/2003/10/Serialization/Arrays"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"/>
</ExportDataResponse>
```

5.3 Fault Codes

The following list of fault codes can be returned when invoking any of the On Key Web Service operations:

Fault Code	Description
SessionExpired	Client session has expired
RecordNotFound	Record requested from the database could not be found
TooManyConcurrentSessions	Client has too many concurrent sessions that have not expired
DatabaseConcurrencyConflict	Concurrency conflict occurred when updating or deleting a record in the database
LogonPasswordExpired	User's password has expired
NotAuthorized	Service operation does not contain a valid <code>SessionId</code> header as authentication token

5.4 List of Services

The complete list of available On Key Web Services can be viewed at [<vdir>/Services/Interfaces/](#). More information on the individual services can be found in the [<it_manual>](#).

6. Change Log

The following sections contain the changes that have been made to the On Key Web Service API for the different On Key releases.

Release	Updates
5.7	New Functionality: - WorkOrderImportService – Ability to Import Work Tasks (Synchronous and Asynchronous) Modifications: - WorkOrderImportService – Import of Work Task Spares has been extended. It now supports all import actions (not only Update) and supports updating all combinations of fields (previously limited to ExternalReference and UserDefinedFields) that can be updated via the On Key user interface. Breaking Changes: - None
5.8	New Functionality: - AssetTaskImportService – Ability to change the Code and Description as part of the import - UserDefinedFieldImportService – Ability to Import User Defined Field Predefined Values (Synchronous and Asynchronous) Modifications: - UserImportService – Import column RequisitionMaximumApprovalAmount added. - UserImportService – Import column ActivePDASessionCode removed. - UserImportService – Import column StaffMemberCode added. Breaking Changes: - UserImportService – Import column WorkOrderMaximumRequisitionApprovalAmount has been renamed to WorkOrderMaximumApprovalAmount.
5.8 SP1	New Functionality: - WorkOrderImportService – Ability to Import Work Order Downtimes (Synchronous and Asynchronous) Modifications: - None Breaking Changes: - SiteService – Import column TimeZoneOffset has been added.
5.9	New Functionality: - AssetImportService – Ability to import multiple GPS points for an Asset - RequisitionImportService – New import service to import: - Requisition Items - Requisition Issues - Requisition Status and Queues

	Modifications: - All On Key business object Ids have been changed from 32-bit integers to 64-bit integers, affecting the following list of import models and model properties of import web services. The namespaces are relative to the root namespace “http://schemas.pragmaproducts.com/onkey” //AssetRegister/v1/ImportAssetComponent.ParentComponentId //Common/v1/ImportAssetTaskAttributeValue.TaskId //Common/v1/ImportAssetTypeTaskAttributeValue.TaskId //Common/v1/ImportAttributeValueDetail.TableId //Common/v1/ImportAssetComponentDocumentLink.ComponentId //Common/v1/ImportDocumentLink.RecordId //Common/v1/ImportDocumentLink.TableId //MaintenanceManager/v1/ImportTask.TaskId //MaintenanceManager/v1/ImportWorkOrderChangeStatusAndQueue.WorkOrderId //MaintenanceManager/v1/ImportWorkTaskSpare.TaskId //MaintenanceManager/v1/ImportWorkTaskSpare.WorkTaskId //SharedConfiguration/v1/ImportUserDefinedFieldPredefinedValue.TableId - The AsyncRequestId for asynchronous imports has been changed from an int-32 to an int-64. The following is a list of asynchronous related web service requests which are affected. The namespaces are relative to the root namespace “http://schemas.pragmaproducts.com/onkey” //v1/FetchAsyncImportProgressRequest //v1.AsyncRequestId //v1/FetchAsyncImportResultsRequest //v1.AsyncRequestId //v1/CancelAsyncImportRequest //v1.AsyncRequestId - The RecordId for import results has been changed from an int-32 to an int-64. This occurs in all synchronous import responses, provided that the Import Record Request’s IncludeRecordSuccesses property is set to true. The namespaces are relative to the root namespace “http://schemas.pragmaproducts.com/onkey” For example: //AssetRegister/v1/ImportAssetComponent.ImportAssetTypeTasksResponse //v1/RecordSuccesses //v1/ImportRecordSuccess - RecordId Similarly, it also occurs in the FetchAsyncImportResultsResponse message //v1/FetchAsyncImportResultsResponse //v1/ImportRecordSuccess - RecordId Breaking Changes: - None
5.9 SP1	New Functionality: - AssetTaskImportService – Ability to do Id based updates for Asset Task Scenario records - AssetTypeTaskImportService – Ability to do Id based updates for Asset Type Task Scenario records - WorkOrderImportService – Ability to do Id based updates for Work Task Labour records Modifications: - SiteImportService – Import column DefaultRequisitionReport added Breaking Changes:

	None
5.10	New Functionality: - AssetTaskImportService – Ability to do import Asset Tasks Sub Tasks has been added. Modifications: - Geographic Data – The ability to import geographic data has been added to the following imports: - Assets - Work Requests - Work Orders - Monitoring Point Readings - WorkOrderImportService – The following Data Members have been removed from the Work Task Spares Used Import. Their values were previously ignored: - AssetCode - ComponentCode - ItemCode - ItemType - ParentComponentCode - TaskCode - WarehouseCode - WorkOrderCode - CalendarImportService – An optional ParentCode Data Member has been added to the Calendar Import. - WorkTaskImportService – An optional Priority Data Member has been added to the Work Task Import. - SiteImportService – Two optional Data Members have been added to the Site Import: DefaultCreditNoteReportCode and DefaultReceiptReportCode. - AssetImportService – An optional Is Linear Data Member has been added to the Asset Import. Breaking Changes: - AssetImportService – On the Asset Import, Data Member GpsData has been replaced with GeographicDataLocation. - CalendarImportService – On the Calendar Import, CalendarType Data Member has been added and must have a value for all import operations which result in a new record being inserted. - StaffMemberImportService – On the Staff Member Import, Calendar Code Data Member has been added and must have a value for all import operations which result in a new record being inserted. - StandardDocumentImportService – On the Standard Document Import the Approved By Data Member has been removed. Its value is now inferred according to the user that the session Id is associated with.
5.10 SP1	New Functionality: - None Modifications: - None Breaking Changes:

	RequisitionImportService – On the Requisition Item Import, Data Members QuantityRequired and ApprovedQuantityRequired have been removed and Data Member Quantity has been added.
5.11	New Functionality: - LocationImportService – Ability to import Staff Location records - UserImportSite – Ability to import User Role Site records - AlarmImportService – Ability to import Alarm records Modifications: - WorkRequestImportService – Ability to import User Defined Fields associated with Work Request records Breaking Changes: - AssetTaskImportService – Ability to import Asset Task Failure Analysis records has been removed to reflect the Failure Analysis updates in On Key.
5.11 SP1	New Functionality: - BudgetTemplateImportService - Ability to import Budget Template records. - BudgetImportService – Ability to import Budget records. Modifications: - None Breaking Changes - None
5.12	New Functionality: - ContactImportService – Ability to import Contacts - LogSheetImportService – Ability to import Log Sheets, Staff Members for Log Sheets, Availability Losses on Log Sheets, Production Batches on Log Sheets, Quality Losses for Production Batches and Performance Losses for Production Batches Modifications: - AssetImportService – Non mandatory columns ShiftSetCode, OperatorListCode, ArtisanListCode and MasterListCode have been added to the Asset Import. - WorkRequestImportService – Non mandatory column OriginatorContactCode has been added to the Work Request Import - WorkOrderImportService - Non mandatory column OriginatorContactCode has been added to the Work Order Import - Work Task Labour imports now support Merges and Inserts. Also, the following non mandatory columns have been added: FinancialYearCode, FinancialYearPeriodCode, StaffToSiteConversionRate and WorkTaskId. - GeneralLedgerCodeImportService - User Defined Fields can now be imported for General Ledger Codes. Breaking Changes: - None
5.12 SP1	New Functionality:

	- None Modifications: - User Defined Fields can now be imported for Material Master and Stock Items. - AssetTypeImportService – The following import services have been removed: - AssetTypeRegularProducts - AssetTypeRegularPerformanceLossReasons - AssetTypeRegularQualityLossReasons - The following columns have been added to the WorkTaskLabourImportService: - WorkTaskId - FinancialYearCode - FinancialYearPeriod - StaffToSiteConversionRate Breaking Changes: - None
5.13	New Functionality - None Modifications: - None Breaking Changes: - None
5.13 SP1	New Functionality: - AssetTypeTaskSubTaskImportService – Ability to import asset type task sub tasks - RequisitionItemImportService – Ability to import new requisition items Modifications: - A Barcode column has been added to the AssetImportService - User Defined Fields can now be imported for Asset Tasks, Locations, Work Order Costing, Suppliers, Sites, Users, and Types of Work - LogSheetAvailabilityLossImportService - import service has been renamed to LogSheetTimeLoss, and the AvailabilityLossReasonCode column has been renamed to TimeLossReasonCode Breaking Changes: - None
5.13.SP2	New Functionality: - ImportAssetTypeComponentAttributeValues method added to AttributeValueImportService – Ability to import Asset Type Component Attribute Values. Modifications: - None Breaking Changes: - None

5.13.SP3	New Functionality • None Modifications: • None Breaking Changes: • None
5.13.SP4	New Functionality • ImportRequisitions method added to RequisitionImportService – Ability to import Requisition headers. Modifications: • ImportWorkOrderCostingItem.AssetName was removed – the value of the asset code on the work order costing item import was never used on the server and therefore removed. This should not cause existing web service consumers to break. • User Defined Fields can now be imported for the following imports on the LogSheetImportService: ○ ImportLogSheets - Log Sheets ○ ImportLogSheetStaffMembers - Log Sheet Staff Members ○ ImportLogSheetProductionBatches - Log Sheet Production Batches ○ ImportLogSheetProductionBatchPerformanceLosses - Log Sheet Production Batch Performance Losses ○ ImportLogSheetProductionBatchQualityLosses - Log Sheet Production Batch Quality Losses ○ ImportLogSheetTimeLosses - Log Sheet Time Losses Breaking Changes: • None
5.13.SP5	New Functionality • None Modifications: • None Breaking Changes: • None
5.14.SP0	New Functionality • TimeAndAttendanceImportService – Ability to import time and attendance records Modifications: • None Breaking Changes: • None

5.14.SP1	New Functionality • None Modifications: • None Breaking Changes: • None
5.15.SP0	New Functionality • ImportAssetTypeTaskRuleLinks method added to AssetTypeTaskImportService – Ability to import Asset Type Task Rule Links. • ImportAssetTypeTaskSpareRuleLinks method added to AssetTypeTaskImportService – Ability to import Asset Type Task Spare Rule Links. • ImportAssetTypeTaskDocumentLinks method added to DocumentLinkImportService – Ability to import Asset Type Task Document Links. Modifications: • A non-mandatory property, InheritChangesToDescendants, was added to the following imports: ○ Asset Type Msi's - AssetTypeMsiImportService.ImportAssetTypeMsis ○ Asset Type Regulars - AssetTypeRegularImportService.ImportAssetTypeRegulars ○ Asset Type Components - AssetTypeComponentImportService.ImportAssetTypeComponents ○ Asset Type Tasks – AssetTypeTaskImportService.ImportAssetTypeTasks ○ Asset Type Task Follow Up Task Links - AssetTypeTaskImportService.ImportAssetTypeTaskFollowUpTaskLinks ○ Asset Type Task Labour Links - AssetTypeTaskImportService.ImportAssetTypeTaskLabourLinks ○ Asset Type Task Scenario Links - AssetTypeTaskImportService.ImportAssetTypeTaskScenarioLinks ○ Asset Type Task Spare Links - AssetTypeTaskImportService.ImportAssetTypeTaskSpareLinks ○ Asset Type Task Special Resources Links - AssetTypeTaskImportService.ImportAssetTypeTaskSpecialResourceLinks ○ Asset Type Task Sub Task Links - AssetTypeTaskImportService.ImportAssetTypeTaskSubTaskLinks ○ Asset Type Task Suppressed Task Links - AssetTypeTaskImportService.ImportAssetTypeTaskSuppressedTaskLinks • On the Asset Type Task Follow Up Task Link import, non-mandatory columns ParentId and FollowUpTaskId were added. AssetTypeTaskImportService.ImportAssetTypeTaskFollowUpTaskLinks • On the Asset Type Task Suppressed Task Link import, non-mandatory columns ParentId and SuppressedTaskId were added. AssetTypeTaskImportService.ImportAssetTypeTaskSuppressedTaskLinks • On the Asset Task Follow Up Task Link import, non-mandatory columns ParentId and FollowUpTaskId were added. AssetTaskImportService.ImportAssetTaskFollowUpTaskLinks • On the Asset Task Suppressed Task Link import, non-mandatory columns ParentId and SuppressedTaskId were added. AssetTaskImportService.ImportAssetTaskSuppressedTaskLinks Breaking Changes:

	On the Asset Task Document Links import, the CopyToWorkOrder column has been renamed to CopyDocumentType and the allowed values are 0: Do not copy, 1: Copy to Work Order or 2: Copy to Work Order Task. DocumentLinkImportService.ImportAssetTaskDocumentLinks
5.16.SP0	New Functionality - None Modifications: - None Breaking Changes: - None
5.16.SP1	New Functionality - None Modifications: - GPS location fields have been added for Site imports Breaking Changes: - None
5.16.SP2	New Functionality - AssetImportService – The ability to import Asset Spares has been introduced - CalendarImportService - The ability to import Calendar Activities has been introduced Modifications: - WorkOrderImportService – ImportWorkOrderCosting: The Description field is now part of the unique key. Breaking Changes: - None
5.16.SP3	New Functionality - LogSheetImportService – The ability to import Log Sheet Time Loss Reasons has been re-introduced Modifications: - None Breaking Changes: - AssetTypeTaskImportService – A new required column, IsNonUsageBasedReadingRequired, has been introduced to the import model for Asset Type Tasks - AssetTaskImportService – A new required column, IsNonUsageBasedReadingRequired, has been introduced to the import model for Asset Tasks

5.18.SP3	New Functionality • AssetTypeTaskSubTaskLinksImportService – The ability to import an Asset Type Task Sub Task has been added
----------	--