FSM ENVIRONMENT AND CONFIGURATION GUIDELINES

Table of Contents

1 FSM ENVIRONMENT MANAGEMENT

This document outlines the standard process to manage all customer's environments. It includes the process of deployment of configurations and modifications from one environment to another during an FSM implementation.

1.1 INSTALLED ENVIRONMENTS

There are three customer environments and two internal environments that are used in new implementations. Each environment has a different purpose and is listed below.

Development

The Development environment is where configurations are created, and unit tested by the project team. Sample data is either created or imported into this environment to test the configuration. Once the configurations and customizations are completed and tested, they are deployed to the Test environment.

All members of the project team (SA, BA, partner/onshore/offshore SWEs) should get access to this environment.

Test

The Test environment is where the project team and customer can test the configurations and customizations promoted from the Development environment. Once all configurations and customizations pass testing, they are to be deployed in the Production environment. If the customer does not have a Migration environment, initial data import is performed and tested on this environment.

Once this environment is created, truncation scripts provided by the R&D should be executed to cleanup demo data.

Production

The Production environment is the "gold" environment for the completed configurations, customizations, and data. All configurations and modifications have passed acceptance testing. This is the source environment that is to be cloned to the Migration environment. All customer data will be migrated into the Production environment prior to going live.

Once this environment is created, truncation scripts provided by the R&D should be executed to cleanup demo data. All basic data needs to be either entered through data migration or manual entry. This environment should not be cloned from DEV or TEST.

Internal Development

The Internal Development environment will be used by GSD software engineers to work on modification development. It is recommended to keep this environment up to date with customer development environment to have the same configurations. If there are any modifications that need code changes and need to use internal environment to develop and build. In this situation we can get the same extract package from customers DEV to TEST and apply to internal DEV.

Internal BNT

The Internal BNT environment is an FSM environment will be used to create and test installation of customer deliveries. This environment will be mainly used by TSM/TL.

If the customer is hosted in IFS cloud it is better to have a Azure environment to test the deliveries as FSM Delivery process is different in On-prem and Azure.

Migration
Currently core FSM only allows 3 environment types DEV, TEST and PROD. But it is possible to create additional environments of type DEV or TEST. The Migration environment should be created as a type DEV to be used in data migration. Migrated data testing where data imports can be tested and verified prior to migration into the production environment. This environment may be cloned to the test environment to provide the test environment with clean, complete data set.

IFS cloud does not allow multiple environments in the same type. For cloud customers we need to use TEST environment for data migration testing.

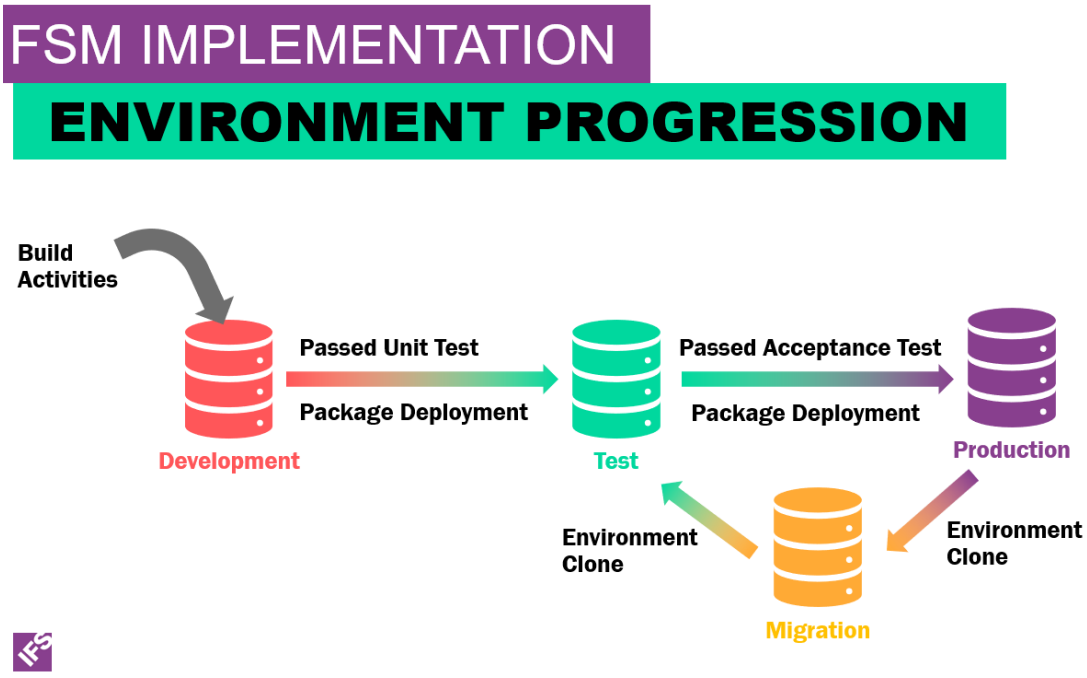
1.2 ENVIRONMENT PROGRESSION

Below is a diagram showing the progression of each environment that a customer may have.

The progression shows all the work completed in the Development environment will be deployed to the Test environment where formal acceptance testing is performed.

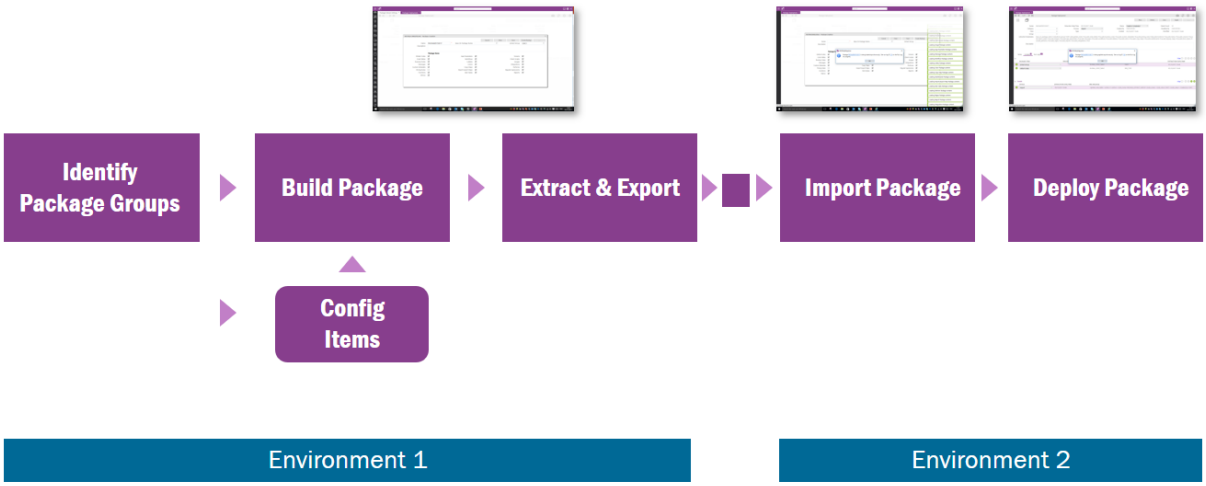
Once it passes, it can be deployed to the Production environment. The configurations and customizations can then be deployed to the Migration environment.

The Migration environment can be cloned to become the new Test environment. The Test environment will then have all the customer data as well as all the configurations and customizations for future testing. This cycle will be repeated multiple times prior to the customer going live.



1.3 PACKAGE DEPLOYMENT

At the beginning of a project, one member from the project team needs to be designated to be responsible for building and deploying packages with one additional member from the team as a backup.



Timing

Configuration package deployment should be done once a week. The day of the week should be determined so everyone on the project team can be prepare for the package build. Packages can also be built and deployed when needed based on the needs of the project. It may be done more frequently for any emergency fixes or less frequently when there are a small number of changes. Should only move the items that are changed from the last package that has been approved. Should avoid moving all configurations each time.

Documentation

Each project team member who has done any configurations or changes to FSM should document details of what was configured or changed. The person delivering the package needs to know every configuration change that must be included. A shared document should be on the IFS SharePoint site or a customer location where the implementation team has access. Each consultant should document what configurations were made.

Business Rules and Process Defs

In the package extract settings for business rules, rule selection is separate from Process Def selection. When including business rules in package deployment, be aware that Custom Process Defs need to be included if they don't already exist in the environment.

Pricing Rules

Pricing rules and financial meta rule searches are included in the package deployment. Packages for Pricing Rules don't include the values from the financial rule and financial list level, requiring a separate export and import on views for PART_FINANCIAL_VIEW and NP_FINANCIAL_VIEW to fully populate pricing rules.

Extract Group Code

Every package deployment should have a unique extract group code defined in global codes under code name PACKAGE_EXTRACT_GROUP. The naming convention should be <customer prefix>_<sequential number> ONLY ie. 01, 02...>. This will provide a log of the configurations and changes with a specific package. A consolidated document of the configurations and changes from each project member will be attached to the package to detail what was included in the package.

This will also be the reference for all the approved configurations and changes included in the package.

Table Scripts

When the table script in custom metadata has a Create statement for a table or view, package deployment will create the table or view. The table script must only include the SQL Create statement.
If the table or view definitions change after they have been initially created, package deployment has no way to alter the table or view. A separate script should be written to alter the table and included with the documented changes so the person moving the configuration between environments can run this through SQL Query tool. Should not drop and recreate any table after initial table. Be aware that all data would be lost if the table is dropped.

1.4 MANUAL DEPLOYMENT

There will possibly be configurations and changes that will not part of a package deployment and will require manual deployment such as table changes and scheduled processes. These items need to be documented in detail and manually created in each environment when approved.

1.5 CODE DELIVERY

Custom code developed by the GCD team will be developed and tested in the Internal DEV environment. Once tested, check in the files to harvest. Technical Solution Manager (TSM) creates a delivery

IFS cloud customers
– TSM create a case to Cloud delivery coordination team.

- CDC team schedule the delivery for initial deployment environment (DEV).
- After testing in DEV, customer schedule installation into TEST and PROD through cloud case.

On Premise customers

- TSM sends the delivery to customer Delivery Recipient.
- Customer installs in DEV
- After testing in DEV, move to TEST and PROD.

1.6 MOBILE CUSTOM DELIVERY

Mobile customizations and configurations will be done by an FSM mobile specialist. A new mobile build will be completed by the mobile specialist and be available for download to the project team. Timing for mobile builds will be determined by the mobile specialist and will vary based on the mobile platform. Apple products will take longer due to the process required by Apple.

Completed mobile source should be checked into Harvest. Instruction document can be found here

2 FSM CONFIGURATION GUIDELINES

The FSM configuration guidelines details the configuration standards during an FSM implementation. This will be the standard practice to manage all configurations and customizations in the customer's environments. Only members of the project team should be allowed to make configurations to the FSM environment. If during the initial discussion with the customer, it is decided that there will be resources from customer also be doing configurations, they should be trained on the process and follow the same guidelines.

4.1 CUSTOMER PREFIX

All configurations will be prefixed with the customer's abbreviation as defined in the Solution Design Document. The abbreviation should be 2-4 characters in length. This is important to identify configurations made by the project team for the customer. All custom metadata naming should begin with the customer prefix, so that it will not be overwritten by any upgrades if the new metadata with the same name is added in a future update.

4.2 CUSTOM PROCESSES/BUSINESS RULES

When a project team is building custom processes or business rules, the following guidelines should be followed.

Reuse Custom Processes if possible, rather than creating new ones for all the rules execution points.

Naming convention for a custom process

- A custom process should be named: <customer prefix>_<table name>_<rule type>_<numeric increment e.g. 01> For example, SWI_PART_NEED_VALIDATION01.
- If a second rule needs to be created, the numeric increment would end in "02". This can occur if a value rule of the execution point or execution type must vary.
- Process Description should be a description about the rule such as "Performs validation on Request".
- Process Name – Copy the process Description, it will be the same, just in upper case.
- Rule Name – Short phrase describing the rule's purpose. The rule name should not be changed once established as it is part the of the record key for package deployment.

Execution point defines where in the sequence of execution the business rule is fired.

Execution Sequence defines the order when process happens in the same table, process type, and execution point. The value should start with 10 with increments of 10. The reason for this is if later we need to add another rule between already defined rules, we do not need to change all rules. Also when defining a new rule, all other rules on the primary table name must be taken into consideration.

Execution type has 4 values which define whether all rules execute or stop at the rule where a specific condition is met. Try to avoid set to "All Rules all Values" as much as possible as this can cause performance issues.

Transactional

Transactional rules execute all rules in the same transaction as the activity that triggers the rule. If an error is encountered, both the activity and rule results are both reversed. The default setting for a

Custom Processes is transactional. Any Custom Process with an after commit execution point cannot be transactional.

When creating custom processes, first select the transactional and synchronous options, except for after-commit rules. For faster response, you can deselect the synchronous option for rules that have no dependencies.

Some Custom Processes require that the triggering transaction be completed before the output transaction can successfully execute where the primary key field of the originating transaction is part of the primary key of the child transaction. Such processes will have to be non-transactional.

From a transactional business process, we'd recommend NOT to execute any XML that forces a new transaction to be initialized.
e.g. a xml api with root element starting with perform_ or mass_update will force another database transaction. Calling this from a transactional business rule process may cause deadlock.

Synchronous

When a Custom Process execute synchronously, it will wait for one business rule to finish before moving on to the next business rule. Custom Processes in a transaction that are all synchronous are all linked together.

The default setting on a Custom Process is not synchronous. Processes execute faster with synchronous unchecked. Business rules typically must be synchronous in order to see changed data updated on a client screen. Synchronous rules are executed in sequence and following changes wait until the rule completes evaluation and applies any changes that result.

Disabling and Deleting Business Rules

Disabling business rules can be done at the process level by the active flag unchecked. This deactivates all rules under the process. Or each individual rule can be disabled by checking the business rule active flag. When the rule is finalized and tested, any disabled rules should be deleted, before adding them to package extraction. Deletion of a business rule, like any metadata deletion, is not recognized by the package deployment tool and requires manual deletion in the target environment.

Value Rules

Business rules with a value process type (value rules) are more efficient than XML rules. Value rules only apply to the same record that triggered the rule. Value rules cannot update a table other than the triggering table. Business rules that update the same record as the triggering record should be value rules rather than XML rules.

Self-Updating XML Rules

Self-updating XML rules should be avoided unless necessary. Try to use a value business rule to set the values on the current record if possible. If it is not possible, only use self-updating business rules as the last resort
One can structure an XML rule that, when triggered by a record change, updates itself, but this is dangerous. It is possible that such a rule becomes recursive, meaning it keeps re-executing in an endless loop. If it becomes necessary for an XML rule to update its own triggering record, the rule inputs need to check for a column that is changed in order to prevent the recursion.
This SQL query will identify self-updating XML rules in the environment.

```
select cpd.process_id,
   prd.rule_sequence,
```

```
    cpd.primary_table_name,
    cpd.execution_point,
    cpd.execution_sequence,
    cpd.execution_type,
    cpd.transactional,
    cpd.transactional,
    ppv.value
from cust_process_def cpd, process_rule_def prd, process_param_perform_val ppv
where cpd.process_id = prd.process_id
    and cpd.process_id = ppv.process_id
    and prd.rule_sequence = ppv.rule_sequence
    and cpd.active = 'Y'
    and prd.active = 'Y'
    and cpd.process_type = 'XML'
    and ppv.value like concat('%<update_',cpd.primary_table_name,'>%')
```

Input Values to Limit Rule Execution

Any business rules under a process that includes any “Update” execution point needs to have rule inputs to limit the execution of the rule to specific data changes. If a rule does not check for some changed data, any update to that record will trigger the rule. Be very specific on what inputs are defined to execute the business rule only when needed. An update rule input must check that a value has changed. This is to prevent excessive business rule execution.

- Avoid using ACTIVITY_FEED feature.
-

4.3 SCREEN CONFIGUATION - UI DESIGNER

When configuring custom screens in the UI Designer, the customization name should begin with the customer’s prefix as defined in the Solution Design Document.

Naming convention: <customer prefix>_<screen name>

Detailed text should be entered into the “Add Comments” dialog box when saving a screen for the configurations that were made. This is very useful when research is needed for any configurations made.

Always use message IDs (Message Translations) and never hard coded labels in the configurations that will be delivered to the customer.

Column configurations should first be defined in Custom Metadata before applying a changed column on the screen. An exception to this is when column configuration must only be visible on a single screen.
Ex: if we add a column to a screen and then add a message id to that column in custom metadata, it will not be applied to the screen. Therefore, add custom metadata with message id first and then add the field to the screen

4.4 INDEX USAGE

When defining custom relations or defining search criteria for screens, consider whether the selects will utilize the correct database indices. if not, consider creating custom database index.

4.5 CUSTOM GLOBAL CODES

All custom global codes should begin with the customer's prefix as defined in the Solution Design Document.

Naming convention: <customer prefix>_<code name>

Any custom global codes should use message IDs. This creates additional setup effort but is easier for handling multi-country requirements. However, baseline global codes do not use message IDs.

Global Code Table is not suitable for storing thousands of values for certain code_name. If you have many code values to work with, consider the use of a custom code table (custom table) instead.

4.6 CUSTOM METADATA

Tables

All custom table names under custom metadata should begin with the customer's prefix as defined in the Solution Design Document. Even though the name is labeled "Table Name", it applies to both table and SQL views.

Naming convention: <customer prefix>_<user defined table name>

Views

The custom metadata for a view must always have the word "view". This makes it easy to identify it as a view and not a table. For views the following naming convention should be followed.

Naming convention: <customer prefix>_<user defined view name>_view

Relationships

All relationship names under custom metadata should begin with the customer's prefix as defined in the Solution Design Document. Relationship ID's should be all UPPERCASE even though the UI allows mixed case.

Naming convention: <customer prefix>_<parent table name>~<child table name>_<relation type ie. join, default, etc.><numeric sequence starting at 01>
Example: ACME_TASK~TASK_STEP_DEFAULT01

If a custom table is created as a child table, there must be a delete relationship on the parent table for this custom child table. This is to prevent orphan records in the database.

Table Script Field

The table script field can be used to automate the creation of tables and views. When the table script in custom metadata has a Create statement for a table or view, package deployment will create the table or view. The table script must only include the SQL Create statement.

User_def Reservation and Descriptions

Before using any user_def column, it is very important to define the purpose of the user_def in custom metadata. Custom metadata is the only place where there is visibility for the purpose of the user_def field. The description field for user_def columns in custom metadata should contain a detail description for the purpose of what the user_def column is reserved for. The column definitions in custom metadata should be the reference for all user_def columns that have been reserved.

Label Messages

The label message ID should be defined on the column definitions for all user_def fields and on all columns on custom tables and views. Custom message translations or baseline FSM message translations would therefore be defined on the message IDs. This provides a framework for easily changing labels and translating labels to other languages.

Column Flags

Max Chars

Defines maximum number of characters in a String column type. Must not be used on any other data types.

Max Decimals

Defines the last position of numbers after the decimal point of a decimal column type. Must not be used on any other data types.

Mask Type

Provides a functional way to limit data types beyond what is defined in the database. For example, a datetime datatype in the database column could be masked to be date only. The mask alters data being saved to the database column and being retrieved from the database column.

Force Select

Force Select deals with the FSM implementation of Optimistic Record Locking.
When working with a record, before anything is done, the record is queried. Values are selected in the query for comparison, a snapshot of those values is made, and the values are kept in memory. For a pending change, just before committing the database change, only the Force Select fields are re-queried and if there are any differences between the new query and the snapshot query, the Optimistic Record Lock is applied. If a field changed, the exception: "Row ({table} {table}, (xxxxxx)) was changed after it was selected. Reselect and try again." occurs, which is known as a "Row Concurrency Error". A field that is not designated as Force Select in the new query is automatically assumed to be null, so if the snapshot query has a value, the Row Concurrency Error will occur.
The reason some fields are not Force Select is that the non-Force Select fields create smaller comparison queries and are therefore more efficient. For example, the baseline FSM metadata configuration of "user_def" fields is set to be non-Force Select so they are excluded in baseline comparison queries, keeping those queries more efficient.

- Set Force Select to Y for any field used in custom metadata, to avoid Row Concurrency Errors.
- Set Force Select to Y for any field that may change from business rules, metadata, policies, integrations, imports and external messaging, but not for fields only changed in the smart client, because the screen manager includes all screen fields in its own comparison queries.
- If a "user_def" is utilized, especially in a business rule, set it to be Force Select in custom metadata.

4.7 CLIENT SCRIPTS

Client scripts provides a means of altering the behavior of screens or data on the screen. A client script only executes in the web client, smart client, or mobile. Client scripts are the closest thing to writing code that is available to metadata configurations. The code in a client script is actually a subset of Java script, therefore some level of coding experience is valuable when configuring client scripts.

Client scripts do not execute from database transactions like business rules do. Instead client scripts execute from screen events.

Naming convention: <customer prefix>_<user defined script name>

To view client scripts guidelines details, press F1 or Go to Help on Client Scripts screen on Smart Client

4.8 CUSTOM FUNCTIONS

Custom functions are automatically generated when a new custom screen is created. By creating a custom function, the custom screen can be added and selected from the menu.

Naming convention: <customer prefix>_<screen name>

4.9 MOBILE SYNC RULES

Mobile synchronization occurs at an interval specified on the Admin screen of the mobile app, for example 1 minute. The mobile app queries the server for new information, which is then delivered to the mobile device based on sync rules.

Mobile sync rules define the data that is transferred to a mobile device and how the data is delivered, whether as information is updated or on a fixed interval. Each sync rule manages one table or view and optionally its parent table or view.

Data can be delivered to specific individuals, for example the task owner, or data can be broadcast to all active users. The delivered information should be limited, or constrained, for example limiting stock information by user or team. This will enable the delivery of only the most appropriate information and minimize data delivery and maximize performance.

Mobile Data Delivery Methods

- Batch-delta – the user specifies the interval in which changes to the table since the last sync are sent to the device. This is generally used for information that doesn't change often, for example Places or Addresses.
- Batch-all – the user specifies the interval in which the entire table is sent to the device. This delivery method is generally not used.
- Real time – when data is changed on the FSM database, it is immediately delivered to the device. This delivery method is generally used for Task information.

Real-Time Rule checklist

- Delivery Method set to Real-Time
- Ownership query returns the person id of the owner. (or Owner checked on Rule) • Owner Identifier populated with the field returned by ownership query (or owner field)
- Related query ONLY if related tables need to be sent with the inserted record.
- Related query should be SPECIFIC to the record that was extracted, in other words if a purchase order is being inserted then it must have a constraint on that specific purchase_order_id to bring in the tables related to that object. It is not just a copy of the INIT query it is constructed specifically to gather related data for a specific record.
- Note: A view cannot be real-time. A real-time rule triggers from the extract and a view has no insert, update or delete so nothing will ever trigger the real-time rule.
- Note: In some cases, a rule may be set to Real-Time and have no queries populated. In this case the data is being initialized on another rule. Commonly, the TASK sync rule fetches a lot

of related data in the initialize query. If a child table has the possibility to return a lot of rows, then it should have its own INIT query to prevent the TASK messages from being too large.

Batch Rule checklist

- Use a delivery method of Batch-Delta whenever possible.
- When using Batch-Delta ensure a modified_dttm field exists on the record.
- Verify a Frequency has been set, typically 1 hour or 24 hours.
- Try to add constraints to the data such as by physical service group.
- Try to define the fields to be updated.
    - Do not use * if large number of records returned.
- Is the max rows attribute large enough to support the number of records?
- Note: When creating server-side database views that will be used in mobile always try to include a modified_dttm column so the view can be configured as a batch-delta rule.

APPENDIX A – SYNC RULE BEST PRACTICE

Plan for periodic review of Sync rule during implementation – not waiting until just before go-live.

Review the sync rules based on the following criteria:

- Is the table being used if not then the sync rule should be deactivated?
  - Do not delete rule as the table may still be needed and populated in a related info query.

- Review proper delivery method (Batch All, Batch Delta, or Real Time)
  - Use BATCH-DELTA whenever possible especially on larger tables.
  - Views need to have a modified_dttm column to use for delta rules. o Views cannot be real-time since there is no extract.
  - STOCK_SERIAL need to be Batch-All otherwise items will never delete from them.
- Review frequency of sync for batch delivery method.
  - Are the constraints on place, product, contact, etc. correct for the business process?
  - If no after-hour/on-demand calls then may not need all the places, etc.
  - Consider disabling the rule and bring in data on TASK related info query. Constrain by physical service group if data is regionalized. o If they have no regions, then possibly remove all constraints (not preferred).
  - Constrain by other records such as place_person relationships (PRIMARY, SECONDARY, FAVORITE, etc.)
  - Possibly use more complicated logic through creation of a view.

- On a real-time rule need to ensure that the ownership query is appropriate.
  - If not doing on-demand/after-hours consider making place, product, etc. a real time rule and find owners by task.

- For large tables (>5K) such as part, product, quality codes, and possibly global codes.
  - Review the max rows limit ensure its large enough. (Default 50K rows.)
  - If possible, find ways to limit number of rows.
  - Only return primary keys and columns (attr) being used in mobile (avoid *).
  - Ensure these are DELTA rules.

- History tables - be sure to constrain them by time or they can get large.

- Verify that all of your sync rules have data constraints.

  Without a where clause, the query issued by a Sync Rule may bring down significantly more data that you expect or need. This is why many of the sync rules have constraints and relationships that use the physical service group A to filter the data.

- Avoid using table_name.* in the attributes (attrs) element of your Sync Rule queries.

  Many times your mobile solution is only going to need a few columns of data from a table, while many tables will have 50, 75, or even well over 100 columns. By specifying just, the columns you need, you can greatly reduce the size of the data sent to the device needlessly. A good example of this is the PART table which has only a few fields defined as attributes.

- Use a max_rows attribute on your Sync Rule queries.

The max_rows attribute guarantees that no matter how poorly a query is filtered, there will be a limit imposed on the amount of data sent to the device. This is especially important when working with new teams or after adding new functionality, where the data might not be setup in a way you expect. A good limit in most cases in 50,000 rows, though for some tables (parts lists, etc.) you may want to set that higher.

- Use a return_only_selected_attrs attribute on your Sync Rules query.

  When you issue hierarchy or tabular selects against the IFS FSM server, it sometimes adds columns to your queries that it thinks you're likely to need. This can result in additional data sent to the device that is unrequired. The return_only_selected_attrs attribute guarantees that you will only get what you request.

- Prefer using Batch-Delta for highly volatile tables with non-time sensitive data.

  Real-Time Sync Rules should be reserved for only the tables where it's critical that a field service engineer get updates as soon as possible. In most cases, syncing a tables changes every hour, several hours, days or even once a week will still support the business processes, and will greatly reduce the volume of messages sent to mobile.

- Prefer using Batch-Delta over Batch – All.

  Batch Synchronization can be set up to only send changes to a table's data, or to send the entire table. Given the efficiency of just sending the changes **Batch-Delta** should be preferred in almost every case.

- Always test your Sync Rule queries using the IFS FSM Client's **XML Poster** screen.

  Available from the Studio\Tools menu, the XML Poster allows you to send the queries associated to Sync Rules to the IFS FSM Server and see the results. This is a good first step to verify that your queries work as expected, and a good final step to ensure that you have the queries filtering the data appropriately.

- Only select data from the parent table in the initial query associated to a Batch-Delta sync rule.

  While Sync Rules allow you to define hierarchy selects that could bring down data from many tables, for Batch-Delta Sync Rules the system will process the data much more efficiently if you only select data from the parent table.

  When you only select data from the parent table, the Sync Rule internally processes the query in pages of database result sets. That increases performance by keeping the individual queries much smaller and greatly reduces the memory footprint of the application while performing these queries. For example, if you were to select 10,000 places and you included child data in that select, the server would select all 10,000 places into memory and then write them in groups of 500 to the messaging service for your devices. If you did not include child data, the server would select 500 rows at a time from the database and then write those 500 to the messaging service for your device.