IBM Integration Bus Message

Modelling Review

IIB version 9, and IIB version 10

Author: Roberto de Gasperi

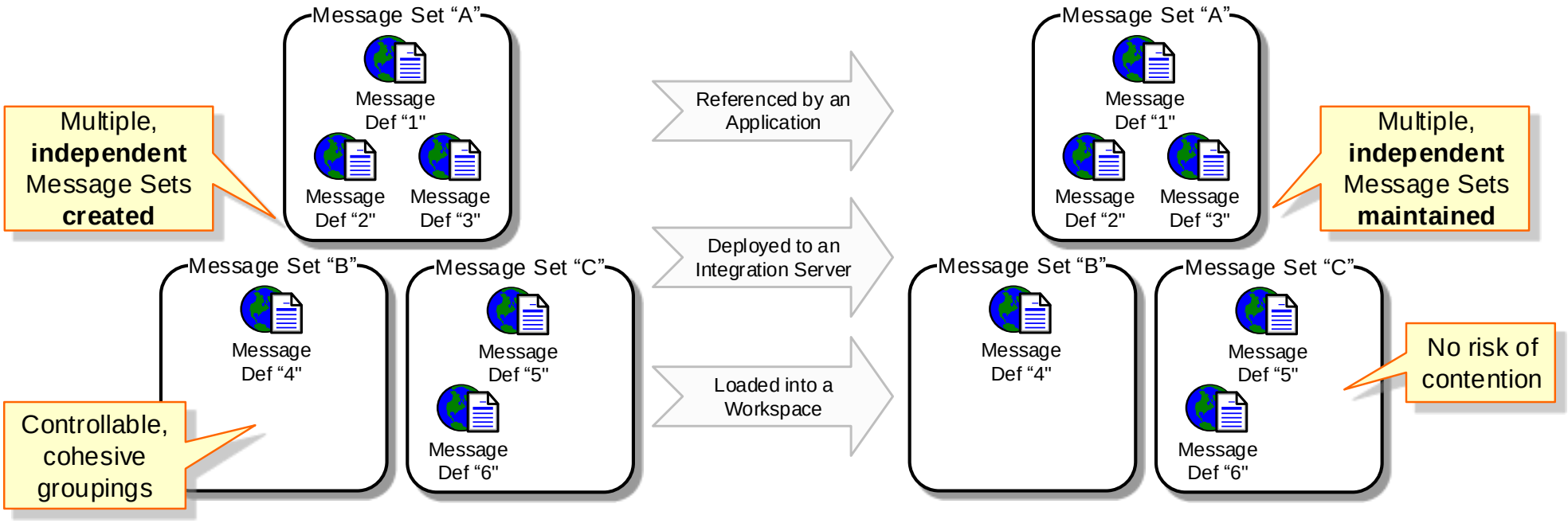
Document Revision: v0.6

Agenda

1. Introduction
    - Problem Statement
    - Message Set Usage
    - Message Model Usage
    - Implementation Comparison: Message Sets vs. Message Models
2. Detailed Walkthrough
    - Use Cases, Issues, and Workarounds
    - Other IIB Message Model Issues?
3. Summary
    - Use Cases, Issues, and Workarounds
    - Root Cause / Solution Analysis
    - Unified / Pooled Message Models
    - IBM Support References

Problem Statement

- Recap of how Message Sets operate… message definitions loaded into one Message Set remain isolated from those held within other Message Sets (at all times)



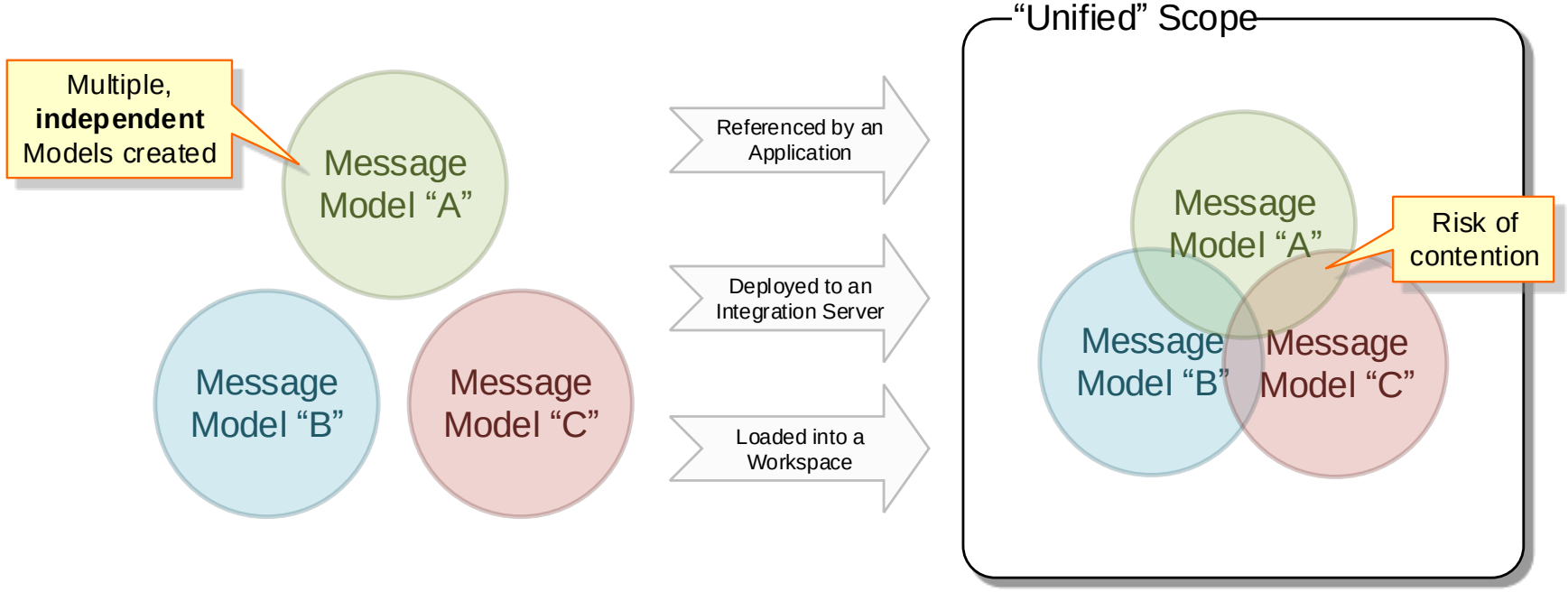
- Integration Developers can choose (design / development time) how, if appropriate, to group message definitions / global elements together within a single Message Set… Taking into account design principles, cohesion, and eliminating any potential contention that might otherwise occur

Problem Statement, continued…

Fast-forward from the days of “Message Broker” and “Message Sets” to a time where “IIB” and “Message Models” are the strategic route forwards…

Problem Statement, continued…

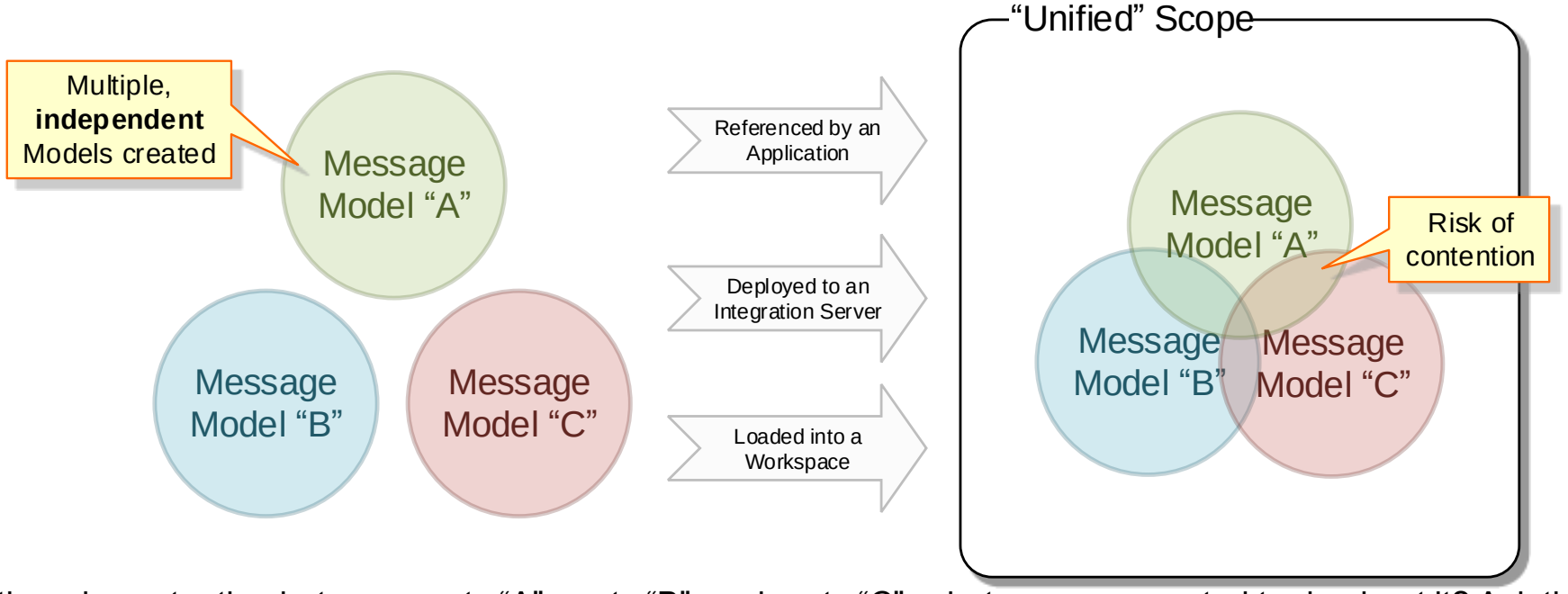
- Excerpt from our in-house Development Handbook:
- "*IIB Message Models; imported into / referenced by an Application, or deployed to an Integration Server as a Library, or loaded into an Integration Toolkit Workspace, are pooled together to create a "unified" Message Model from all models in-scope. This creates an environment in which multiple forms of contention can occur*"



…for us, this represents an **inferior** Application Architecture. Contention is occurring amongst; our message definition artefacts, those supplied to us from third-party integration partners, and those built-in / supplied by IBM!

Problem Statement, continued…

- In some respects, this presentation should end here, it has presented enough of an issue already… if Message Models “A”, “B”, and “C” represent models for different messaging parties, different Organisations, under **no circumstances** should their message definitions be amalgamated together
- This is an **anti-pattern for integration**



…if there is contention between party “A”, party “B”, and party “C”, what are we expected to do about it? Ask the parties to change their native message definitions because of a product limitation within the integration layer?!

Question / Follow-up?

- Of course, luck might dictate that you “get away” without any contention occurring between unrelated Message Models, but putting this aspect aside, even without contention issues…
- **Are there any Use Cases whereby the lack of isolation between multiple Message Models actually helps?**

Message Set Usage

- Historically, Message Sets were used to house message definitions for:
  - XML-based interfaces, including SOAP-XML
  - SAP interfaces, a discovery process utilising the IBM SAP Adapter
  - Bespoke message formats, fixed-width / tag-delimited record structures etc (MRM Domain)
- “Message Sets” are defined within a “Message Set Project”
- A “Message Set Project” is “referenced” from Message Flow “Projects” (Integration Projects, Applications etc) which need to use them, each “Project” supporting multiple “Message Flows”
  - i.e. a dependency model is established through Eclipse project referencing
- Note: Message Sets continue to be supported for…
  - Legacy MRM scenarios – to ensure backwards compatibility of existing interfaces
    - There is (currently) no migration tool for MRM to DFDL, attempting a conversion risks uncovering a whole host of defects / functionality gaps / product issues
  - New requirements for which the issues faced with Message Models cannot be overcome / worked-around

Message Model Usage

- Message Models are used to house message definitions for:
  - XML-based interfaces, including SOAP-XML
  - SAP interfaces, a discovery process utilising the IBM SAP Adapter
  - Bespoke message formats, fixed-width / tag-delimited record structures etc (DFDL Domain)
- A direct replacement for Message Sets – or so we thought / had hoped!
- You may choose to define “Message Models” within “Libraries”
  - Enabling their reuse across multiple code projects (Integration Projects, Applications)
  - This choice absolutely, materially impacts your response to the “product” issues you will encounter…
    - It doesn’t lead to / cause any of the issues, but it has made some of the workarounds possible and others easier to manage
- A Message Model “Library” is “referenced” from Message Flow “Projects” (Integration Projects, Applications etc) which need to use them, each “Project” supporting multiple “Message Flows”
  - i.e. a dependency model is established through Eclipse project referencing – just like with Message Sets!

Implementation Comparison

- Key aspect: Integration Developers must explicitly name the Message Set / message type that they expect to be used by their interfaces (parser properties / message tree meta-data properties)
    - Being explicit is great – it unlocks a robust world whereby:
    - Interfaces receiving data are explicit about what data they expect to receive
    - Interfaces producing data are explicit about what data they expect to produce
    - There exists a high degree of transparency as to which message type selections are intended at development time, plus efficient message type selection / matching at runtime
- When implementing Message Models within an Application or Static Library, there is no longer a requirement to name the model and / or message type expected (inbound, message receivers) / intended (outbound, message producers)
    - The list of available Message Models (for message type selection / matching) are **ALL** available global elements from within **ALL** Message Models in-scope
    - Less transparent at development time – which Message Model and / or message type is intended? There may be a whole list in-scope
    - Less efficient at runtime – again, the product must search the list of Message Models / global elements to find any match from all of those in-scope

Question / Follow-up?

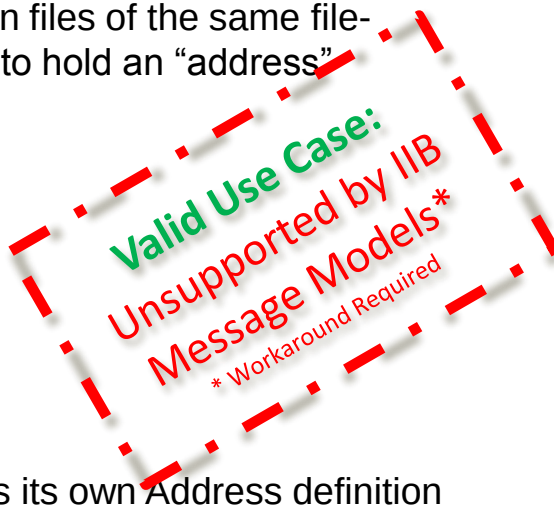
- There are methods / techniques of overriding Message Set selection dynamically at runtime (if necessary) e.g. MCD headers for MQ, re-routing within Message Flows and using in-line validation nodes etc
- Even with the lack of explicit Message Model / message type selection, you must STILL reference the correct subset of Message Models (if externally housed in a Library) or import the correct subset of message definitions if housed in-line within the same project to ensure that the intended Message Model is in-scope
    - i.e. the lack of explicitness is in any case a half-way house, not holistically applied, clearly you must know what message definitions you are developing towards
- **Are there any Use Cases whereby the (semi) lack of explicitness with regards to Message Model / type selection actually helps?**

Use Cases, Issues, and Workarounds

- The slides that follow depict the typical Use Cases for which we have encountered implementation issues when using Message Models, and the workarounds we have adopted to counter these issues
  - Slides have been grouped / arranged into numbered themes (1), (2) etc
- Common to all issues / Use Cases is:
  - We're not attempting to do anything unusual / weird / wacky here, we're simply attempting to work with message definition artefacts (like many, typical IBM Customers would do?)
    - …if you think that we are doing something unusual, please call it out!
  - The Use Cases can be **wholly fulfilled**, and **without workaround**(s) when implementing Message Sets
    - All issues are due to the way that IIB has evolved
- We are still discovering more issues the longer we work with IIB!

Use Cases (1)

- Use Cases:
  - 1a) Support multiple message definitions generated from a SAP system (i.e. for various RFC functions)
  - 1b) Support multiple message definitions supplied from various third-parties
- Additional context:
  - Multiple message definition packages can feasibly contain message definition files of the same file-pathname. To illustrate this example, consider two Message Models created to hold an “address” message definition for two different Organisations:
  - {IIB Library}:OrganisationA_Address_Model
    - <root directory>\Address.xsd
  - {IIB Library}:OrganisationB_Address_Model
    - <root directory>\Address.xsd
  - …both Message Models contain an “Address” file, but each Organisation has its own Address definition (perfectly feasible / reasonable), the Address definition for Organisation A is different to the Address definition for Organisation B but both are named “Address.xsd” and both are position in the root of their respective message definition packages



Issues (1)

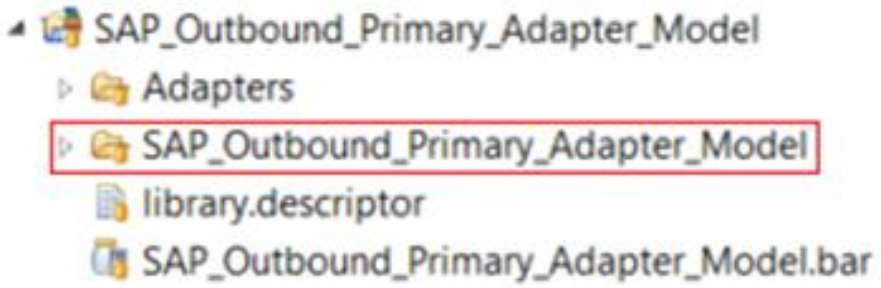
- If any difference is found between two XSDs with the same Message Model file-pathname, across multiple Message Models an IIB “error” arises (two XSD files of the same file path, but a different definition, cannot be referenced by a single IIB Application or deployed to the same Integration Server)
- This restriction / limitation does not comply with how XSDs are managed in the real-world, or according to XML standards
    - XSD objects are governed by namespaces – they need only be unique within a namespace, different organisations adopt / implement organisation-specific namespaces for this reason
    - The file-pathname of an XSD should not be significant
- Why is this issue arising?
    - When an IIB Application attempts to create a “unified” Message Model by combining all Message Models that are referenced by an Application / deployed to an Integration Server, IIB cannot resolve why the contents of two message definition file are different (despite the XML namespace of the two files being different!)

Issues (1)

- Additional issue factors:
  - As a result of this issue, IIB / Message Models are less W3C / XML standards-compliant than their predecessor – IBM Message Broker / Message Sets
  - An integration platform should be flexible to interface provider's requirements, and should not need to ask them to re-package or change their (perfectly valid) message definitions due to an integration tooling limitation
    - Integration solutions should be transparent, flexible, and compliant with established messaging standards for its supported protocols

Workarounds (1)

- **Manually define** a Message Model **sub-folder** for all Message Model Libraries as-standard, **manually move** all imported / created message **definitions files** beneath this sub-folder.…
  - All message definition files within a Message Model Library must be stored within a folder / sub-folder named after the Message Model itself, for example:
  - This ensures that message definition files do not contend with one another, for example, if message definition files of the same file-pathname are encountered in multiple message definition packages (Message Model Libraries)
- Note: This mimics the level of isolation offered by Message Sets, whereby Message Sets (akin to a Message Model folder) sit within a Message Set Project (akin to a Message Model Library)
- Message Model Sub-folder Examples (left: Non-SAP, right: SAP)…



Workarounds (1)

- Revisiting our earlier example…
  - {IIB Library}:OrganisationA_Address_Model
    - OrganisationA_Address_Model\Address.xsd
  - {IIB Library}:OrganisationB_Address_Model
    - OrganisationB_Address_Model\Address.xsd
- …the file-pathname of the two “Address.xsd” files are now different, so the filename contention that occurs when a “unified” Message Model is derived is resolved
- Note: Due to the inconsistent nature of the various “New > Message Model” wizards, the creation of a sub-folder and movement of definition files may need to be completed manually
  - First define an erroneous Message Model
  - Then fix-up the erroneously generated Message Model structure by hacking around with the filesystem structure of the Project itself

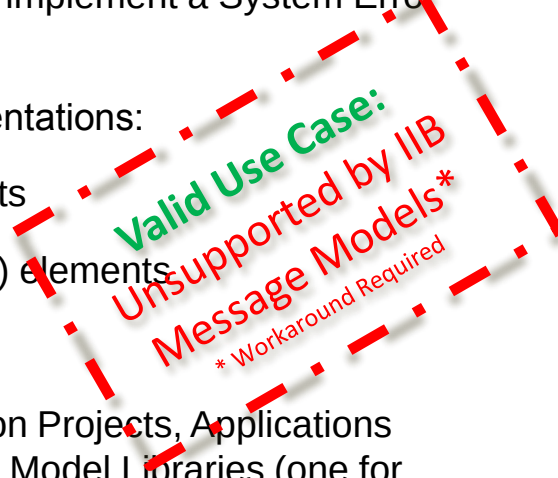
Use Cases (2)

- Use Cases:
  - 2a) Define a Proxy-Service that re-advertises a Service Method of a Sub-ordinate (SOAP)
  - 2b) Define a Composite-Service that re-implements a Service Method of a Sub-ordinate (SOAP)
  - 2c) Define a Composite-Service that interacts with one or more Sub-Ordinates (SOAP)
  - 2d) Define multiple Message Flows to implement different Operations of a Service definition (SOAP)
- Additional context:
  - When importing WSDLs into a Message Model, IIB “selectively" adds 'ibmSchExtn' prefixes to the global elements within the associated messaging XSDs. It does so in order to identify which global elements are in-fact referenced within the WSDL definition
    - Presumably allowing IIB to be more efficient at runtime – having a “short-list” of searchable elements to perform message type selection against?
    - However, the reason for using these extensions has not been clarified since a Support Ticket suggested removing them altogether, thus suggesting that they are not required by the Product (see PMR 45919,001,866)

Use Cases (2)

- For example, an XSD message element that **is not** referenced by a WSDL:
  - `<xsd:element name="BusinessErrorFaultResp"`
- For example, an XSD message element that **is** referenced by a WSDL:
  - `<xsd:element ibmSchExtn:docRoot="true" name="SystemErrorFaultResp"`

- Additional context:
  - The previous example demonstrates how a particular Service definition may implement a System Error Fault message, but not a Business Error Fault message (for example)
  - …there may be a whole set of mixed scenarios for different Service implementations:
    - Some Services that implement all available message (global) elements
    - Some Services that implement a subset of available message (global) elements
    - Some Services which share some message elements, but not others
  - In order to fulfil the Use Cases depicted, Message Flow “Projects” (Integration Projects, Applications etc) must implement multiple Message Models / reference multiple Message Model Libraries (one for each SOAP-WSDL package / Service definition)



Issues (2)

- If any difference is found between two XSDs with the same Message Model path and file name, across multiple Message Models an IIB “error” arises (two XSD files of the same file path, but a different definition, cannot be referenced by a single IIB Application or deployed to the same Integration Server)
- …this is a **copy + paste** from the previous Use Cases / Issues – the same root cause!
- What is additionally ridiculous / frustrating for this particular issue is that the conflict between two like-for-like message XSD files was created by IIB itself!
    - The XSDs supplied to IIB were consistently implemented for the same XSD file-pathname and namespace definition, it is the product itself that is selectively extending one and not another XSD file, then complaining to the user that the two are no longer consistent
    - IBM is tripping-up over its own product limitations here
- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, coupled this time with the lack of explicit Message Model / type selection driving the need for IBM Schema extensions to be introduced in the first place

Issues (2)

- Additional issue factors:
  - Given that message definitions are held within Message Models using an XSD format, and for these depicted Use Cases the Customer's message definitions are already held within XSDs…
  - There should be no need for IIB to edit / manipulate / extend / hack-around with the XSD files presented to it
  - …especially if it is then going to complain that the two like-for-like files are no longer consistent!

Workarounds (2)

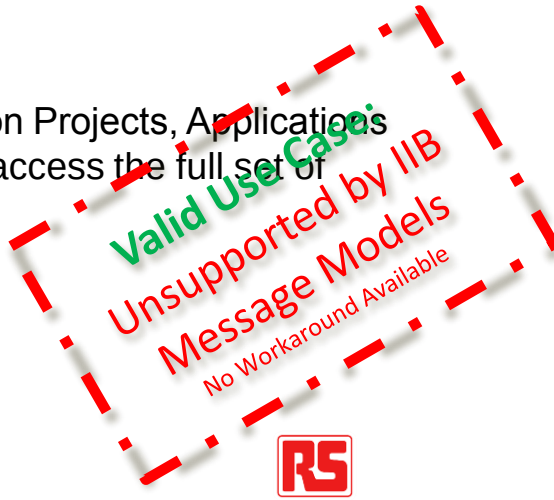
- **Manually add** any missing ‘‘ibmSchExtn’ declarations to all global message elements within each message XSD after each time that a Message Model is generated, or updated, from a WSDL package
  - Doing this as-standard prevents contention if the same messaging XSD appears in multiple Message Models that are referenced from an IIB Application
  - …but negates the benefits case for introducing the IBM schema extension in the first place
- Revisiting our earlier example…
  - An XSD message element that **is not** referenced by a WSDL:
    - `<xsd:element ibmSchExtn:docRoot="true" name="BusinessErrorFaultResp"`
  - An XSD message element that **is** referenced by a WSDL:
    - `<xsd:element ibmSchExtn:docRoot="true" name="SystemErrorFaultResp"`
- …both carry the extension, thus both global elements are in-scope for message type selection purposes, contention will not occur for the current Message Flow (as a Service) or future, Composite Service Scenarios

Use Cases (3)

- Use Cases:
  - 3a) Split / breakdown SOAP Service definitions across / into multiple WSDL files
  - 3b) Extend a WSDL file with another (SOAP) e.g. in order to specify additional endpoints
  - 3c) Implement a third-party WSDL package which includes multiple WSDLs for communicating with a third-party Web Service
- Additional context:
  - Multiple WSDLs can be defined for the same Business Service and so, in the case where WSDL files are named after their respective Business Service, they may have the same filename
    - Perfectly valid, since each WSDL has a distinct namespace to differentiate them from one another
    - A different namespace results in different storage path / package path (a different sub-folder) within the WSDL package therefore filesystem integrity is maintained
      - Multiple files, same filename, different pathname
  - …remember, filenames are not significant in XML, namespaces are significant

Use Cases (3)

- Additional context:
  - For example, a primary WSDL fully defines a Business Service boundary:
    - <root directory>\external\service\customer\v0100\CustomerService.wsdl
    - `targetNamespace="urn:external:service/customer/v01.00"`
    - `… <service definitions etc> …`
  - Whilst, a secondary WSDL extends the primary to include additional (non-publishable) endpoint information
    - <root directory>\internal\service\customer\v0100\CustomerService.wsdl
    - `targetNamespace="urn:internal:service/customer/v01.00"`
    - `… <endpoint details etc> …`
  - In order to fulfil the Use Cases depicted, Message Flow “Projects” (Integration Projects, Applications etc) must implement / import / reference the extended WSDL file in order to access the full set of Service definition details



Issues (3)

- IIB no longer preserves / respects either the namespace package path, or the file-system hierarchy of the WSDL file(s) within a WSDL package when importing into a Message Model
    - Albeit IIB does preserve / respect the file-system hierarchy of any associated XSDs from the same package
    - Instead, IIB simply places WSDL files at the root of the Message Model
        - …refactoring import references accordingly i.e. editing / transforming the WSDL contents (perfectly valid to begin with)
    - Respecting any hierarchy will suffice, and negate the need for automatically refactoring the file – but IIB does not do this
- In the Use Cases above, this causes file-system contention during the import process – two files of the same name cannot be placed in the same location (obviously!)
- Secondary to this, the presence of more than one WSDL file of the same filename cannot be supported **across an IIB estate / Integration Server scope / Workspace Project scope**
    - Contention occurs between Message Models when a “unified” Message Model is derived

Issues (3)

- Why is this issue arising?
  - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model exacerbates the impact of this issue, coupled this time with the fact that IIB Message Sets respect the namespace hierarchy of WSDL and XSD files, importing files into a package structure according to their target namespace definition whilst Message Models do not
  - The migration path from Message Sets to Message Models does not offer full equivalence
  - Once again, as a result of this issue, IIB / Message Models are less W3C / XML standards-compliant than their predecessor – IBM Message Broker / Message Sets

~~[Workarounds]~~ Alternatives (3)

- No IIB Message Model workarounds available – external solution required to deal with the product limitation…
- Do not import more than one WSDL of the same name into IIB across the whole Integration estate
  - **Manually refactor** / remaster supplied WSDLs according to this IBM product limitation
  - Or, consider importing the base WSDL only, **discarding the extension** definitions
  - Or, **revert to using Message Sets**, IIB Message Models do not suffice

Future Solutions (3)

- In future, we have been informed that IIB will re-label conflicting WSDL files on-import, applying an increasing index number / suffix to the filename (PMR 46338,001,866)
- For example, if there are multiple CustomerService.wsdl files in the scope of a single import, then these will be named as CustomerService1.wsdl, CustomerService2.wsdl etc within the target Message Model folder
- This is a poor solution from the following perspectives:
  - It does not address the underlying cause of the issue – that a WSDL package, perfectly valid from a W3C WSDL specification perspective, needn't have its resources relocated, then its references refactored in order for it to remain valid in the context of an IIB Message Model
  - Also, this solution does not negate the possibility that CustomerService(n).wsdl may appear in two different IIB Message Models, with different contents – depending upon the order in which the numeric identifiers are allocated for different imports
    - Due to the "unified" Message Model contention, IIB will error if two like-for-like file-pathnames have differing contents across the current scope (of Project references, or deployed Integration Server artefacts)

Use Cases (4)

- Use Cases:
  - 4a) Implement multiple XSD packages that contain XSD files for which no target namespace is defined
- Additional context:
  - The use of namespaces may be recommended, but they are not mandatory in XML
  - Some third-parties define XSD packages which are not namespace-qualified, we must implement integration solutions for their interfaces
    - Large, global companies supply XSDs in this way
  - Multiple message definition packages can feasibly contain message definition files without a target namespace defined, supplied by different Organisations
    - Different Organisations (integration partners) are not, should not, and need not be aware of one another… their interfaces may not even be related
    - An element name e.g. “firstName”, “addressLine”, “town” etc may easily co-exist within multiple packages, with a different, Organisation-specific definition

Use Cases (4)

- Additional context:
  - To illustrate this example, consider two Message Models, one created to hold a “widget” message definition for one Organisation and one created to hold a “gizmo” definition for another Organisation:
  - {IIB Library}:OrganisationA_Widget_Model
    - <root directory>\Widget.xsd
  - {IIB Library}:OrganisationB_Gizmo_Model
    - <root directory>\Gizmo.xsd
  - …both Message Models contain XSD files without a target namespace defined, each file-pathname is different but each XSD happens to have an element named “objectId”
    - For Organisation A, an “objectId” is a 10 character field
    - For Organisation B, an “objectId” is an 8-digit number

Issues (4)

- Although the file-pathnames may be different (avoiding filename contention within IIB Message Models), because IIB attempts to derive a “unified” Message Model from all Message Models in-scope i.e. all those referenced by a single IIB Application or deployed to the same Integration Server, element-contention occurs where XSDs do not implement a unique / Organisation-specific target namespace
    - IIB is forcing XML definitions to be clubbed-together / amalgamated within the same data domain / realm (the default namespace domain / realm) when in fact they have no awareness of one another and so a process of de-duplication has not occurred across the multiple Organisations
- This restriction / limitation does not comply with how XSDs are managed in the real-world, or according to XML standards
    - XSD objects are governed by namespaces yes – but namespaces are **not** mandatory
    - Unrelated XSD packages need not be amalgamated in this way
        - They can be kept separated from one another, as they are within Message Sets – avoiding this issue entirely

Issues (4)

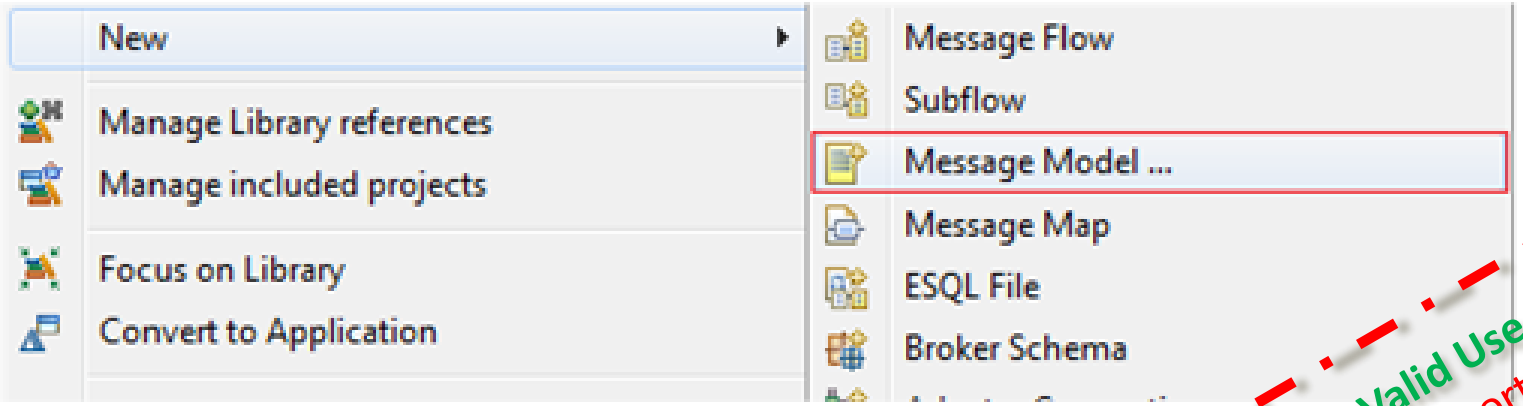
- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model
- Additional issue factors:
    - Once again, as a result of this issue, IIB / Message Models are less W3C / XML standards-compliant than their predecessor – IBM Message Broker / Message Sets
    - Once again, an integration platform should be flexible to interface provider’s requirements, and should not need to ask them to re-package or change their (perfectly valid) message definitions due to an integration tooling limitation
        - Once again, integration solutions should be transparent, flexible, and compliant with established messaging standards for its supported protocols

~~[Workarounds]~~ Alternatives (4)

- No IIB Message Model workarounds available – external solution required to deal with the product limitation…
- **Revert to using Message Sets**, IIB Message Models do not suffice
  - When working with XSDs that do not have a target namespace defined, a Message Set (not a Message Model) must be generated
  - Message Sets offer an additional level of isolation (each Message Set is fully independent of one another) over Message Models (which are not isolated from one another at build-time or runtime)
- Using Message Models as an interim solution for these circumstances is not advantageous and can simply delay the discovery of contention, causing rework of new and possibly existing Message Models

Use Cases (5)

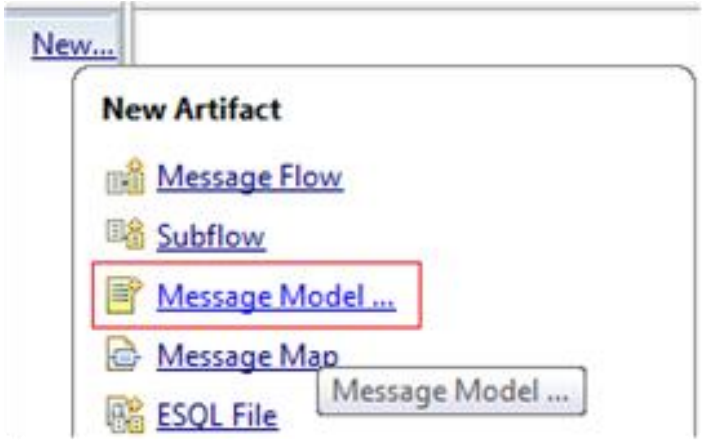
- Use Cases:
  - 5a) Define an XSD-based Message Model within a Project (Application, Library etc) according to a package structure / hierarchy, following and maintaining accurate import references (XSD file dependencies)
- Additional context:
  - The IIB Toolkit contains multiple access paths to defining a new Message Model
    - E.g. right-click on a Project and select “New > Message Model…” from the context menu



Valid Use Case:
Unsupported by this IIB Method*
* Workaround Required

Use Cases (5)

- Additional context:
  - E.g. use the New Artefact context menu and select “New Artifact > Message Model…”



Valid Use Case:
Unsupported by this IIB Method*
* Workaround Required

Issues (5)

- By default, XSDs are simply placed at the root of the resultant Message Model using the previously illustrated New > Message Model methods (above)
- Imported / referenced XSDs are not followed, resulting in a broken Message Model to begin with
    - Multiple New > Message Model steps must be repeated for each XSD within a package
- This restriction / limitation is a major inconvenience, again, not reflecting how XSDs are managed in the real-world (not all XSDs are simple, flat, single-file “hello world” samples)
- Why is this issue arising?
    - When Message Models were introduced as a concept, dedicated options were added to the Message Model creation wizards to cater for generation-by-XSDs
        - As per the previous “New > Message Set” wizards
    - However, it seems that there is a lack of consideration for what XSD management entails, plus inconsistencies were introduced into the Integration Toolkit functionality…

Issues (5)

- Additional issue factors:
  - There is another Message Model initialisation method which assists with these Use Cases, ironically, by behaving **completely inconsistently** with the previously illustrated methods (above)
  - From a user-perspective, each method of generating a Message Model from the same source should result in:
    - The same functionality being executed
    - And certainly – the same end result being obtained
    - There is certainly nothing within the tooling / accompanying documentation to suggest otherwise
    - Why should one “New > Message Model” option be different to another?
  - Even this additional method is not perfect, and comes with further limitations of its own
    - IBM themselves are confused about this, as our many PMR requests have demonstrated, we’re left to find our own, lucky workarounds…

Workarounds (5)

- **Manually define** the required sub-folder structure during the import process
  - Each folder within an XSD package must be hand-cranked
  - Ensure the original filesystem structure is maintained, else this will result in broken import-links
  - Repeat the New > Message Model process and manual steps for each XSD
  - …This can be a painstaking activity, folder-by-folder, XSD-by-XSD
  - …Any manual processing errors result in a broken / invalid Message Model, references are not refactored on-import, XSDs references may be missed
- Alternatively…

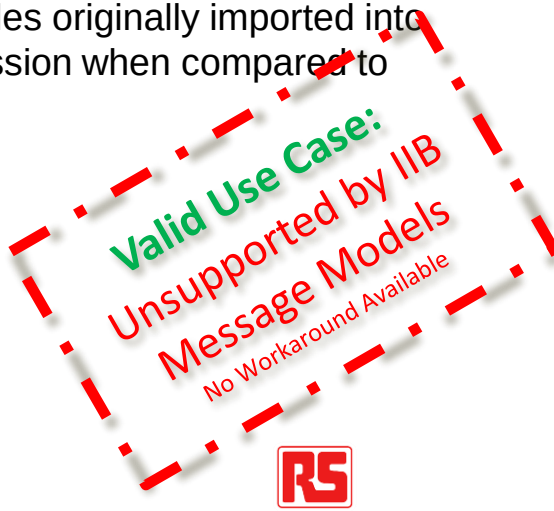
Workarounds (5)

- Use the **Quick Start** context menu and select “Quick Start > Start from WSDL and/or XSD files…”

  - Use of this “Quick Start” wizard requires the Message Model sub-folder workaround, from the earlier slide “Workarounds (1)”, to be applied **after** the initial Message Model generation
    - Remember, all message definition files must be manually moved on the file-system to beneath this Message Model sub-folder
    - Watch out, we now have one Workaround interacting with another!
  - This Message Model generation method comes with its own limitations…

Workarounds (5)

- Although this method will import dependent XSDs in a reproducible manner, it won't preserve the complete file-system hierarchy of the full XSD package imported
  - Instead, only the sub-portion of the file-system structure required to preserve the relative path references between the import files is generated within the Message Model structure
  - E.g. If there are (currently) no dependent XSDs, then a flat structure is generated within the target Message Model irrespective of its original file-system structure / location of the XSD being imported
  - …Obviously, if a new level of sub-folder is later introduced into the hierarchy of an XSD dependency chain e.g. when updating an existing Message Model from an updated XSD package, then different hierarchy will be generated when the Message Model is regenerated / updated
    - This will lead to broken version history for the individual XSD files originally imported into the model – this is not ideal, and represents a functional regression when compared to Message Set (*.mxsd file) behaviour

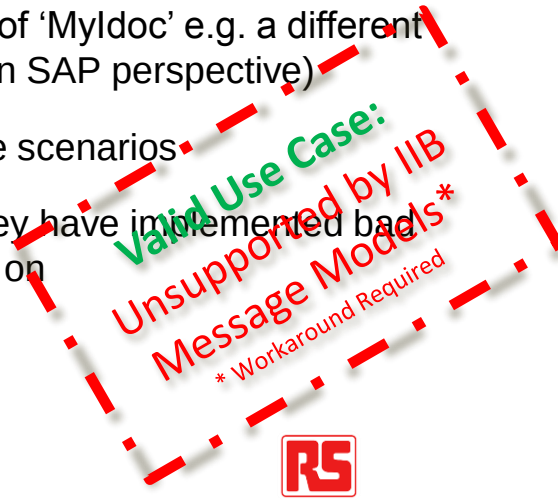


Workarounds (5)

- This quite literally is a case of picking between the lesser of multiple evils…
  - Use the standard wizards – manually do battle with them
  - Or, use the “Quick Start” wizard – more practical, but be aware of its limitations also
  - Combine the two methods (hybrid technique)…
    - Use the “Quick Start” wizard for first-time / new Message Models only
    - Use the “New > Message Model” wizard for small-scale updates to existing Message Models
    - Even with this hybrid technique – accept you will lose some Source Control version history / links to historical files when updating existing Message Models and a filesystem hierarchy change is triggered because of the “Quick Start” wizard

Use Cases (6)

- Use Cases:
  - 6a) Use default discovery settings for generating Message Models for **multiple SAP IDocs** using the IBM SAP Adapter and the IBM SAP Adapter Discovery Wizard
- Additional context:
  - Consider the following example where **two Message Models** are required for the **same IDoc type** discovered from **two different SAP applications**:
  - Message Model Library 1 “SAPCRM_MyIdoc_Model” – generated from the SAP CRM implementation of the ‘MyIdoc’ IDoc
  - Message Model Library 2 “SAPERP_MyIdoc_Model” - generated from the SAP ERP implementation of the ‘MyIdoc’ IDoc
  - …where each SAP application has a slightly different view / implementation of ‘MyIdoc’ e.g. a different extension applied (this may not be good practice but allowable / valid from an SAP perspective)
    - As integration teams, we must still build integration solutions for these scenarios
    - We cannot refuse to integrate another system because we believe they have implemented bad practice (internal to their application), it is not our system to comment on



Issues (6)

- If you import the depicted Message Model Libraries into a Workspace linked to the same Application, or attempt to deploy them to the same Integration Server, **two errors** occur…

    1. "Global type xxxx exists more than once within the Application … Exists in: /SAPERP_MyIdoc_Model/….xsd /SAPCRM_MyIdoc_Model/….xsd"

    - …Essentially, IIB is not happy that a Message Model file, contains the same namespace-qualified element(s) across two Message Models linked to the same Application, but with a different definition

    2. "The file path '….xsd' exists more than once within the Application and its referenced libraries but contains different content. This file path can exist more than once within all projects that are part of the Application, but the files must have the same content"

    - …Essentially, IIB is not happy that an "….xsd", has the same file-pathname across two message models linked to the same Application, when the contents of those files are different

- IBM did not consider how (SAP) Message Models may be used / implemented in the real-world for IDocs
- With Message Sets (as opposed to Message Models), each message definition is entirely isolated from any other Message Set

    - Previously – the contents of one Message Set did not interfere with another, thus the SAP inconsistency depicted, which is not an issue for SAP itself, was not an integration issue either

Issues (6)

- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
    - SAP IDoc discovery used to be a simple, out-of-the-box procedure, that worked successfully across:
        - Multiple SAP implementations (different Organisations)
        - Multiple SAP applications (CRM, ERP)
    - These Message Model issues compromise the effectiveness of the IBM SAP Adapter with IIB
        - A key, strategic capability in justifying IIB’s use over rival integration products in an SAP environment
    - In the worst case, users don’t see the potential for Message Model contention early-on in and get tied in knots with multiple points of contention / having to rework existing integration assets
        - Can lead to inconsistent artefacts, with a mixture of workarounds applied
    - There is a lack of clear guidance and documentation of the pitfalls within the IIB product documentation

Workarounds (6)

- **Multiple workarounds** are required to prevent SAP Message Model contention from occurring within the IIB **Integration Toolkit** and at **runtime** for SAP IDocs
  - The workaround below is required in addition to other e.g. series (1) i.e. Use Cases (1), Issues (1), Workarounds (1) discussed earlier, which are generally applicable to all IIB Message Models implemented…
- **Business Object Namespace Customisation**
  - The standard “Business object namespace:” property value of ‘`http://www.ibm.com/xmlns/prod/websphere/j2ca/sap`’ must be customised according to the source / target SAP application
  - Such customisation was not previously required when using Message Sets – one Message Set-per-IDoc (no contention occurred between Message Sets)
  - Customising this value ensures that any message elements are uniquely attributed a specific SAP application, avoiding contention where two message elements are defined using the same name, but two (slightly) varying definitions between multiple SAP applications
  - Note: This namespace has to be **further customised** to be IDoc-specific if similar / further message element contention occurs between **two different IDocs** for the same SAP application

Workarounds (6)

- Revisiting our earlier example…
  - Message Model Library 1 “SAPCRM_MyIdoc_Model” – generated from the SAP CRM implementation of the ‘MyIdoc’ IDoc
  - Message Model Library 2 “SAPERP_MyIdoc_Model” - generated from the SAP ERP implementation of the ‘MyIdoc’ IDoc
  - …both Message Models can now co-exist, regardless of implementation discrepancies between SAP CRM and SAP ERP applications for the “MyIdoc” object
    - “MyIDoc” for SAP CRM has been assigned a CRM-specific namespace
    - “MyIDoc” for SAP ERP has been assigned an ERP-specific namespace
  - **Plus** – other Message Model workarounds have been applied e.g. defining a Message Model-specific sub-folder to house message definition files of a similar name (preventing further IIB Message Model file contention)

Use Cases (7)

- Use Cases:
  - 7a) Use default discovery settings for generating Message Models for **multiple SAP BAPIs** using the IBM SAP Adapter and the IBM SAP Adapter Discovery Wizard
- Additional context:
  - Consider the following example where **two Message Models** are required for an **RFC (BAPI) of the same name** discovered from **two different SAP applications**:
  - Message Model Library 1 “SAPCRM_MyBapi_Model” – generated from the SAP CRM implementation of the ‘MyBapi’ RFC
  - Message Model Library 2 “SAPERP_MyBapi_Model” - generated from the SAP ERP implementation of the ‘MyBapi’ RFC
  - …where each SAP application has a slightly different view / implementation of ‘MyBapi’ e.g. as simple as a different documentation comment applied to a single element within SAP (this may not be good practice but allowable / valid from an SAP perspective)
    - As integration teams, we must still build integration solutions for these scenarios
    - We cannot refuse to integrate another system because we believe they have implemented bad practice (internal to their application), it is not our system to comment on

Issues (7)

- If you import the depicted Message Model Libraries into a Workspace linked to the same Application, or attempt to deploy them to the same Integration Server, **two errors** occur…
    1. “Global type xxxx exists more than once within the Application … Exists in: /SAPERP_MyBapi_Model/SapZstpWsMyBapi.xsd /SAPCRM_MyBapi_Model/SapZstpWsMyBapi.xsd”
    - …Essentially, IIB is not happy that a Message Model file, contains the same namespace-qualified element(s) across two Message Models linked to the same Application, but with a different definition
    2. “The file path '….xsd' exists more than once within the Application and its referenced libraries but contains different content. This file path can exist more than once within all projects that are part of the Application, but the files must have the same content”
    - …Essentially, IIB is not happy that an “….xsd”, has the same file-pathname across two message models linked to the same Application, when the contents of those files are different
- IBM did not consider how (SAP) Message Models may be used / implemented in the real-world for BAPIs
- Once again, with Message Sets (as opposed to Message Models), each message definition is entirely isolated from any other Message Set
    - Previously – the contents of one Message Set did not interfere with another, thus the SAP inconsistency depicted, which is not an issue for SAP itself, was not an integration issue either

Issues (7)

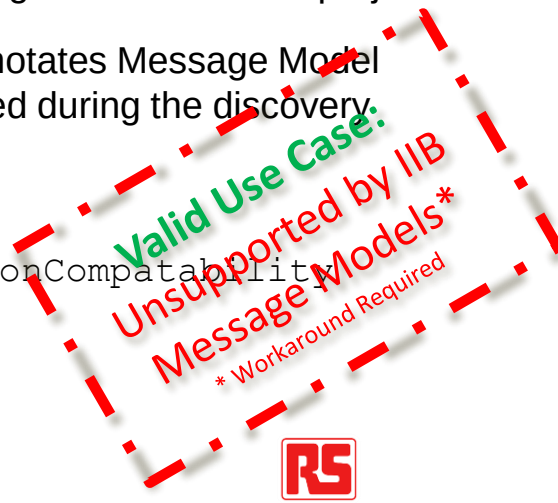
- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
    - SAP BAPI discovery used to be a simple, out-of-the-box procedure, that worked successfully across:
        - Multiple SAP implementations (different Organisations)
        - Multiple SAP applications (CRM, ERP, BW)
    - These Message Model issues compromise the effectiveness of the IBM SAP Adapter with IIB
        - A key, strategic capability in justifying IIB’s use over rival integration products in an SAP environment
    - In the worst case, users don’t see the potential for Message Model contention early-on in and get tied in knots with multiple points of contention / having to rework existing integration assets
        - Can lead to inconsistent artefacts, with a mixture of workarounds applied
    - There is a lack of clear guidance and documentation of the pitfalls within the IIB product documentation

Workarounds (7)

- **Multiple workarounds** are required to prevent SAP Message Model contention from occurring within the IIB **Integration Toolkit** and at **runtime** for SAP BAPIs
  - The workaround below is required in addition to other e.g. series (1) i.e. Use Cases (1), Issues (1), Workarounds (1) discussed earlier, which are generally applicable to all IIB Message Models implemented…
- **Business Object Namespace Customisation**
  - The standard “Business object namespace:” property value of ‘`http://www.ibm.com/xmlns/prod/websphere/j2ca/sap`’ must be customised according to the source / target SAP application
  - Such customisation was not previously required when using Message Sets – one Message Set-per-BAPI (no contention occurred between Message Sets)
  - Customising this value ensures that any message elements are uniquely attributed a specific SAP application, avoiding contention where two message elements are defined using the same name, but two (slightly) varying definitions between multiple SAP applications
  - Note: This namespace has to be **further customised** to be BAPI-specific if similar / further message element contention occurs between **two different BAPIs** for the same SAP application

Use Cases (8)

- Use Cases:
  - 8a) Perform a patch upgrade of an IIB Integration Node / Toolkit or apply an Interim Fix to the IBM SAP Adapter for an existing Integration estate
- Additional context:
  - A minor patch upgrade e.g. an Interim Fix applied to the IBM SAP Adapter should not require all existing integration assets to be rediscovered from SAP for IDoc Message Models
    - Backwards compatibility should be ensured between fix-levels
    - This has always been the case in the past (when using Message Sets)
  - The effort involved with / regression impact of… full-scale rediscovery across an integration estate e.g. circa 50 IIB Message Models for SAP IDocs may be too great to justify an upgrade / as a minor project
  - IIB, utilising the IBM SAP Adapter for SAP message definition discovery, annotates Message Model XSDs with meta-data attributes e.g. the version of the Resource Adapter used during the discovery process:
    - `<info:resourceAdapter xmlns:info=http://www.ibm.com/xmlns/prod/j2ca/versionCompatability version="<edit this token>"/>`



Issues (8)

- Following a minor upgrade, if you **rediscover an existing** (any single) Message Model for an SAP IDoc, or attempt to **discover a new** (even just one) Message Model for an SAP IDoc, if you link the Message Model to an Application or deploy the Message Model to an Integration Server containing other, existing SAP IDoc Message Models, an **error** occurs…

    1. "The file being processed was ...**SapIDocControlRecord.xsd** in library <Model>. The message from the XMLNSC validator is: sch-props-correct.2: A schema cannot contain two global components with the same name this schema contains two occurrences of 'http://www.ibm.com/xmlns/prod/websphere/j2ca/sapr/sapidoccontrolrecord,SapIDocControlRecord'."

    - …Essentially, IIB is not happy that a Message Model file (the IDoc Control Record XSD), is not consistent across multiple (i.e. the new and existing) Message Models

    - The IDoc Control Record is a standard structure present for all IDocs, located within every Message Model for SAP IDocs. The only difference between the new and existing Message Models are changes introduced by IBM itself – the Resource Adapter version attribute increment

- Once again, with Message Sets (as opposed to Message Models), each message definition is entirely isolated from any other Message Set

    - Previously – the contents of one Message Set did not interfere with another, thus the version number was allowed to be tracked accurately, and independently for each Message Set without causing any contention between Message Sets. It was a useful item of meta-data, and not problematic for IIB

Issues (8)

- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
    - This issue is an example of Message Model contention occurring **within IIB** and **because of IIB** itself!
        - This contention is not user-influenced… a user object has not changed, or been introduced to cause the contention, this issue is not caused by a “style” choice
        - It appears that IBM has tripped over it’s own Message Model (lack of) isolation issues
    - Once again, this new implication is not clear within the product documentation
    - This issue can apply to other message definition files initialised during the discovery process i.e. other XSDs representing the definition of other IDoc records / segments
        - If an IDoc segment / XSD appears in multiple IDoc structures, this will result in another shared message definition file appearing in two Message Models with different content, and the same contention will occur
        - Shared segments are common across IDoc structures

Workarounds (8)

- If an upgrade has occurred, the following additional workaround is required to be applied to SAP IDoc Message Models in order to prevent contention, or avoid future contention from occurring
  - Once again, this is in addition to all other applicable Message Model workarounds for this scenario
- **Cleanse the SAP Resource Adapter Version**
  - Repeat the following procedure for any SAP IDoc Message Model file / XSD for which the resource adapter version annotation causes contention…
  - **Manually** edit the standard / generated 'SapIDocControlRecord.xsd' to amend the Resource Adapter version attribute:

```
<xsd:appinfo
  source="http://www.ibm.com/xmlns/prod/j2ca/versionCompatability">
 <info:resourceAdapter
  xmlns:info=http://www.ibm.com/xmlns/prod/j2ca/versionCompatability
  version="<edit this token>"/>
</xsd:appinfo>
```

  - Set the token value to a fixed (arbitrary) value, or a value reflecting the existing Resource Adapter versions of Message Models already deployed on the target integration platform

Workarounds (8)

- This is a poor (but necessary) workaround for multiple reasons:
  - Users should not need to edit the contents of discovered / application generated artefacts – the discovery process should ensure that the resultant message definition is accurate, supportable, and can be repeatedly obtained with success
    - Future resources may need to maintain / rediscover the Message Model – they need clarity on how the structure was obtained in the first place
  - The version attribute loses its meaning, but worse still – could be misleading
    - What version of the SAP Adapter was used to generate edited / hacked / worked-around Message Models? This context is lost
- Alternatively… always **rediscover**, **recompile**, **redeploy**, **retest** ALL Message Models pertaining to SAP IDocs and related code artefacts each time that an IBM SAP Adapter is upgraded
  - This might sound like a sensible Continuous Integration / Delivery strategy in some contexts, but it's the wrong justification for covering-up a product issue
  - This involves mass rework (rediscovery risks introducing additional changes), not just redeployment!
  - The upgrade in question may be as small as applying a focused, Interim Fix to the IBM SAP Adapter component of IIB – can every such initiative pay incur the cost of mass **re**work?

Use Cases (9)

- Use Cases:
  - 9a) Deploy a change to an existing, shared XSD within an XSD package / Message Model without impacting all (other) Message Models that reference that shared XSD
  - 9b) Introduce a new Message Model based upon an XSD package containing a shared XSD without impacting existing Message Models that reference that shared XSD
- Additional context:
  - The same XSD file / namespace qualified XML element may reused across multiple XSD packages, or referenced by multiple messaging XSDs within an XSD package
    - XSD packages may be supplied by third parties, and/or derived from recognised standards etc
  - Consider the following example:
  - {IIB Library}:Invoice_Model
    - <root directory>\InvoiceMessages.xsd, Address.xsd
  - {IIB Library}:Order_Model
    - <root directory>\OrderMessages.xsd, Address.xsd

Use Cases (9)

- Additional context:
  - …both Message Models contain an “Address” file / definition which are consistent with one another
  - …both Message Models are derived from two XSD packages, an “Invoice.zip” XSD package, and an “Order.zip” XSD package (or could be derived from the two messaging XSDs within the same package)
    - The “Address.xsd” file is pertinent to both an Invoice message and an Order message since both contain addressee details, which are consistent with each other – thus they make use of a **shared** address schema to define their respective messaging XSDs
  - It’s useful to note that the Address.xsd may be mastered within an XSD repository, but packaged multiple times / as-needed, so that each package can be published independently of each another
    - That doesn’t mean to say that the Address.xsd is not shared, in this example it is! Both Address.xsd files are derived from the same source, they are the same file, they just happen to be packaged-up into two distinct packages / referenced by two distinct messaging XSDs
  - From an integration perspective, if XSDs packages are supplied in this way, representing two distinct interfaces, they and their related code assets also need to be flexibly managed, independently
    - It is important to implement the correct level of granularity when translating published artefacts into code units / assets for change-management purposes

Issues (9)

- In terms of initial deployment, there is no issue…
  - Both resultant Message Models can be deployed independently, to the same Integration Server if desired, and although a “unified” Message Model will be derived / constructed in-memory, no conflict will arise (according to the current example)
- In terms of future maintenance, issue arise, consider the following scenario…
  - A change is introduced to the Address definition and it is deemed to be a fix to the current version
  - From a change-management perspective it is deemed necessary to rollout the change to the Order interface independently of the Invoice interface, a phased-rollout
  - Problem (1)
    - Even working with Message Flows as **Independent Resources**, we cannot redeploy the Order Message Model (containing an updated Address.xsd) whilst the Invoice Message Model is also deployed (containing a reference to the old Address.xsd)
    - Error: “A schema cannot contain two global components with the same name this schema contains two occurrences of xxxx”
    - We are locked-in, two independent Message Model Libraries tied-at-the-hip at deployment time

Issues (9)

- Problem (2)
    - Working with Application containers would not suffice either, it simply moves the point of conflict from a deployment-time conflict / issue, to a development-time conflict / issue… the updated Order Message Model cannot be referenced by the same Application as the yet to be updated Invoice Message Model
- Problem (3)
    - Even if we want to update both the Invoice Message Model and the Order Message Model at the same time – we cannot rollout the changes as “updates”
    - In a deployment sequence e.g.
        - Deploy updated Message Model Library: Invoice Model
        - Deploy updated Message Model Library: Order Model
        - Deploy each updated Message Flow (Independent Resources)
    - Once again, the first Message Model cannot be redeployed whilst the second is deployed
        - The first “mqsideploy” attempt will always fail “A schema cannot contain two global components with the same name this schema contains two occurrences of xxxx”

Issues (9)

- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
    - It’s interesting to note from these examples how IIB **Independent Resources** are not so independent
        - They would be independent but for the “unified” Message Model approach
    - We totally lose the ability to manage the two Message Models / two interfaces independently
    - Even if granular change control is not required, we still have to cope with additional deployment complexity
        - When deploying one Message Model component, you must always maintain an overall view of the contents of all other Message Models i.e. the opportunity for conflict
    - None of these restrictions were true of Message Sets, none of these issue would have arisen
    - This kind of evolution doesn’t seem like progress from a user perspective(?)
        - It’s far more restrictive, and doesn’t help

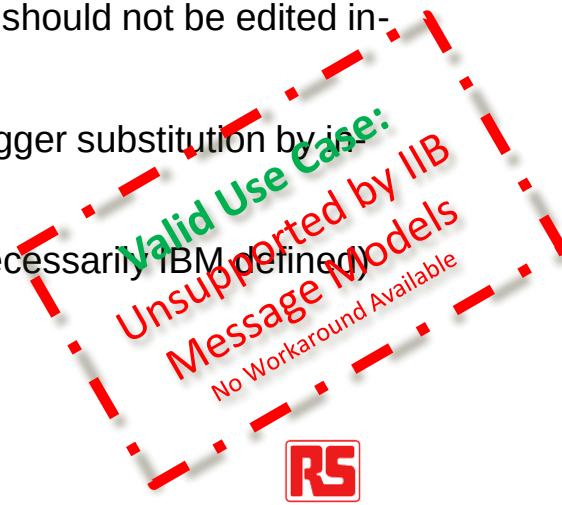
~~[Workarounds]~~ Alternatives (9)

- No IIB Message Model workarounds available – external solution required to deal with the product limitation…
- Amend the technical implementation to workaround the product limitation, one of:
  - Change the **deployment topology** so that the conflicting Message Models no longer need to reside within the same Integration Server together
  - Change the **design / structure** of the Application / Message Flows so that the conflicting Message Models no longer need to reside within the same Application container
- Amend the **change management approach** and rollout the changes to all conflicting Message Models / related Message Flows at once
  - For Independent Resources, this workaround additionally requires a change to the deployment approach / sequence...

1. All **Message Flows** referencing any impacted Message Models must be **stopped**
2. All impacted **Message Models** must be **deleted** (where currently deployed)
3. Deployment of updated components can commence, then restart stopped components

Use Cases (10)

- Use Cases:
  - 10a) Utilise (any) Source Control Keyword Substitution within message definition files (WSDLs, XSDs)
  - 10b) Work with SOAP version 1.2 Message Models when using Subversion (SVN) for Source Control
- Additional context:
  - Customers may wish to add Source Control Keywords to message definition files
    - E.g. version / revision numbers might be added to an XSD annotation in order to allow auto-tracking of Source Control meta-data within the message definition artefact
  - Integration teams might be supplied message definition files which contain Keywords that trigger substitution by in-house Source Control repositories
    - E.g. standard message definition files, or third party defined files that should not be edited in-house
  - IBM themselves may supply “IBM defined” message definition files which trigger substitution by in-house Source Control repositories
    - A reserved “IBMdefined” folder is used to house IBM-supplied (not necessarily IBM-defined) message definition files within Message Models



Issues (10)

- When Keyword Substitution occurs within one Message Model, **substituted keywords cause contention** where the same message definition files appear within other Message Models (e.g. reused / shared / common XSDs), **errors occur**…
    - Remember - two IIB Message Models cannot contain the same file unless their contents are identical, otherwise errors occur at development-time (due to Application Project references) or at deployment-time (for Independent Resources)
- Also, it may seem ridiculous, but yes, IBM supply “IBM defined” message definition files within a dedicated Message Model folder called “IBMdefined” which are not compatible with the way that Message Models work
    - Once again, IBM is tripping-up over its own product limitations here…
- The SOAP version 1.2 XSD is supplied by IBM within SOAP version 1.2 Message Models (“IBM defined”)
    - This XSD contains a comment block which includes a Subversion (SVN) Keyword

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Schema defined in the SOAP Version 1.2 Part 1 specification
      Recommendation:
      http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
      $Id: soap-envelope.xsd,v 2099/01/01 09:00:00 jbloggs Exp$
```

Issues (10)

- Each time that the respective Message Model is committed to an SVN repository that recognises this as a keyword to be substituted, the keyword value is amended…

```
$Id: soap-envelope.xsd,v 2099/12/12 10:10:59 jdoe Exp$
```

  - Contention occurs between this Message Model and all other SOAP version 1.2 Message Models referenced by the same Application or deployed to the same Integration Server

- Why is this issue arising?
  - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
  - Integration teams should inherit / use, not edit; supplied third party, industry standard e.g. SOAP, or IBM defined message definition files
  - This restriction was not true of Message Sets, this issue would not have arisen
  - It is not possible to simply edit the “value” of the keyword-back, because each time the edited version is re-committed to Source Control, its keywords are substituted once again and the contention reoccurs
  - The presence of substitutable keywords within **industry standard files** reinforces the Use Case

~~[Workarounds]~~ Alternatives (10)

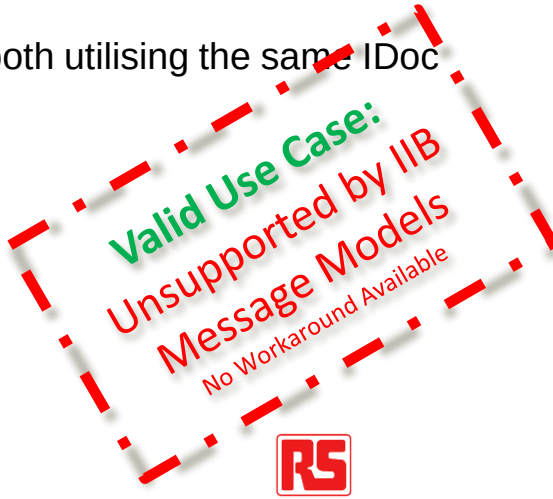
- No IIB Message Model workarounds available – external solution required to deal with the product limitation…
- Don’t use Keyword Substitution within in-house defined message definition files
- **Revert to using Message Sets**, IIB Message Models do not suffice
    - When working with supplied message definition files / industry standard message definition files / IBM supplied files e.g. SOAP version 1.2 (not to be edited) which contain substitutable keywords, a Message Set (not a Message Model) must be generated
    - Message Sets offer an additional level of isolation (each Message Set is fully independent of one another) over Message Models (which are not isolated from one another at build-time or runtime)
- Using Message Models as an interim solution for these circumstances is not advantageous and can simply delay the discovery of contention, causing rework of new and possibly existing Message Models

Future Solutions (10)

- In future, IBM are working on a solution whereby IBM defined files are centrally located on the IIB Integration Node Server (installed with the product)
    - …And referenced from there at runtime, “avoiding” Message Model contention for IBM supplied / defined files
- This is a poor solution from the following perspectives:
    - Actually this solution avoids the real issue, it is more akin to an workaround (albeit IBM implemented)…
        - The real issue here is the “unified” Message Model / a lack of Message Model isolation, which is impacting a large number of issues, not just these Use Cases
    - This solution only deals with “IBM defined” message definition file issues, issues with Customer files remain unaddressed
    - Lose development-time reference from within the Integration Toolkit, referenced files from the Integration Node Server are only referenced at runtime
- Initial solutions provided by IBM have not worked successfully
- There is no commitment on when a fix will be completed – the Support Ticket for this issue has not received a response for over 12 months (Legacy PMR: 72799,001,866 / Case Id: TS00003859)

Use Cases (11)

- Use Cases:
  - 11a) Implement two different SAP Release Versions of the same SAP IDoc
- Additional context:
  - SAP is versioned, like many other systems
    - Supporting transitioning / pinning-back of resources (for backwards compatibility purposes following upgrades) / concurrent versions (for impact / change management)
  - Some SAP processes may utilise newer versions of an IDoc than others
    - E.g. there may be a requirement to support a legacy IDoc message format for SAP Release Version '46C', at the same time as a more up-to-date message definition for the same IDoc at SAP Release Version '702' (for different Business processes)
  - Customers may need to support two interfaces to SAP for IDoc integration, both utilising the same IDoc type, but for different SAP Release Versions:
    - {IIB Library}:SapDelvry03Zdelvry1_IDoc_**V46C**_Model
    - {IIB Library}:SapDelvry03Zdelvry1_IDoc_**V702**_Model



Use Cases (11)

- Additional context:
  - When utilising the IBM SAP Adapter for IDoc discovery purposes, the SAP IDoc “type” / variant dictates the name of the generated top-level message definition files within the resultant IIB Message Model
    - E.g. “`SapDelvry03Zdelvry1.xsd`” where “`Delvry03`” represents the IDoc Type and “`Zdelvry1`” represents the CIM Type (extension)
    - Note: This is not a behavioural change with the IBM SAP Adapter itself (bundled with IIB), this has always been the case (Message Broker too), and it makes sense for the IBM SAP Adapter to do so
  - Also, the IBM SAP Adapter adds a meta-data token to generated message definition files, tracking the SAP Release level that was used during discovery (a form of audit trail for Customers / IBM Support)
    - E.g. “`<sapasi:IDocVersion>46C </sapasi:IDocVersion>`” or “`<sapasi:IDocVersion>702 </sapasi:IDocVersion>`”

Issues (11)

- A conflict is caused within IIB – where the same IDoc needs to be supported at two different SAP Release Versions (a basic / classic requirement to support concurrent versioning)
- The rest is pretty much a copy + paste from issue / theme (6), but for reasons similar to issue / theme (8) – a hybrid problem…
- If you import the depicted Message Model Libraries into a Workspace linked to the same Application, or attempt to deploy them to the same Integration Server, **two errors** occur…
    1. "Global type xxxx exists more than once within the Application … Exists in: / SapDelvry03Zdelvry1_IDoc_**V46C**_Model/….xsd / SapDelvry03Zdelvry1_IDoc_**V702**_Model/….xsd"
    - …Essentially, IIB is not happy that a Message Model file, contains the same namespace-qualified element(s) across two Message Models linked to the same Application, but with a different definition
    2. "The file path '….xsd' exists more than once within the Application and its referenced libraries but contains different content. This file path can exist more than once within all projects that are part of the Application, but the files must have the same content"
    - …Essentially, IIB is not happy that an "….xsd", has the same file-pathname across two message models linked to the same Application, when the contents of those files are different
- Once again, IBM did not consider how (SAP) Message Models may be used / implemented in the real-world for IDocs, when utilising it's own, existing Adapter technology

Issues (11)

- With Message Sets (as opposed to Message Models), each message definition is entirely isolated from any other Message Set
    - Previously – the contents of one Message Set did not interfere with another, thus the SAP inconsistency depicted, which is not an issue for SAP itself, was not an integration issue either
- Why is this issue arising?
    - Once again copy + paste from the previous Uses Cases / Issues – the “unified” Message Model, exposing one Message Model Library to the contents of another, creating the potential for conflict
- Additional issue factors:
    - This issue is an example of Message Model contention occurring **within IIB** and **because of IIB** itself!
        - This contention is not user-influenced
        - Even if the top-level of an SAP IDoc has not changed between two release versions, contention still occurs due to meta-data conflicts (“IDocVersion” tag)
        - Once again, it appears that IBM has tripped over it’s own Message Model (lack of) isolation issues
    - Once again, this new implication is not clear within the product documentation

~~[Workarounds]~~ Alternatives (11)

- No IIB Message Model workarounds available – external solution required to deal with the product limitation…
- Do not reference two concurrent versions of a SAP IDoc Message Model from within the same IIB Application, or attempt to deploy them to the same Integration Server
  - **Amend** the Integration **solution design** to allow for IIB Application or Integration Server separation (if appropriate)
    - Customers should not need to track such product-inflicted conflicts
    - It may not be possible / appropriate to amend the solution design, what if two conflicting Message Models need to be referenced from the same Message Flow?
  - Or, **revert to using Message Sets**, IIB Message Models do not suffice

A Brief Pause…

- All of the previous issues are true for IIB version 9 and IIB version 10 alike…
- During previous IBM workshops we were led to believe that IIB version 10 might offer some improvements in Message Model isolation levels, in comparison to IIB version 9

Additional IIB Version 10 Considerations…

Shared Libraries within IIB Version 10 onwards have re-implemented the isolation levels offered by IIB Message Sets:

- Shared Libraries do not expose the contents of other Shared Libraries to one another
- Developers must explicitly choose / define a Shared Library to interact with at runtime
    - In-fact, the existing ‘MessageSet’ property within the Broker / IIB Properties tree is used for this selection

However, there are a few issues here:

- Shared Libraries are not as supportive as Static Libraries e.g. they do not currently support working with Adapters
- They come with “sharing” at runtime i.e. deploying a change to a Shared Library at runtime impacts all Applications referencing that Shared Library
    - A consequence that may not be desirable (greater levels of impact analysis required, wider regression implications, principles of share-nothing etc)

This appears to be an admission of fault (with Static Libraries), the initial and only Library implementation available prior to IIB Version 10, otherwise Shared Libraries would work in much the same way as Static Libraries.

So – why not back-port this aspect of Shared Libraries to Static Libraries and Application Message Models, allowing Customers to choose an appropriate implementation, free of product limitations?

Additional IIB Version 10 Considerations…

So, we're left with a somewhat confused product offering…

Shared Libraries don't offer the same functionality level / support as Static Libraries e.g. Adapter support, but do offer greater levels of isolation for Message Models than Static Libraries, making them easier to work with, but they are "shared" and as such come with sharing, which in other ways makes them less isolated!

Or, to look at it another way, **Shared Libraries** offer **no sharing** (pooling) **between Message Models**, Static Libraries do… whilst **Shared Libraries** offer **sharing at deployment / runtime**, whilst Static Libraries do not.

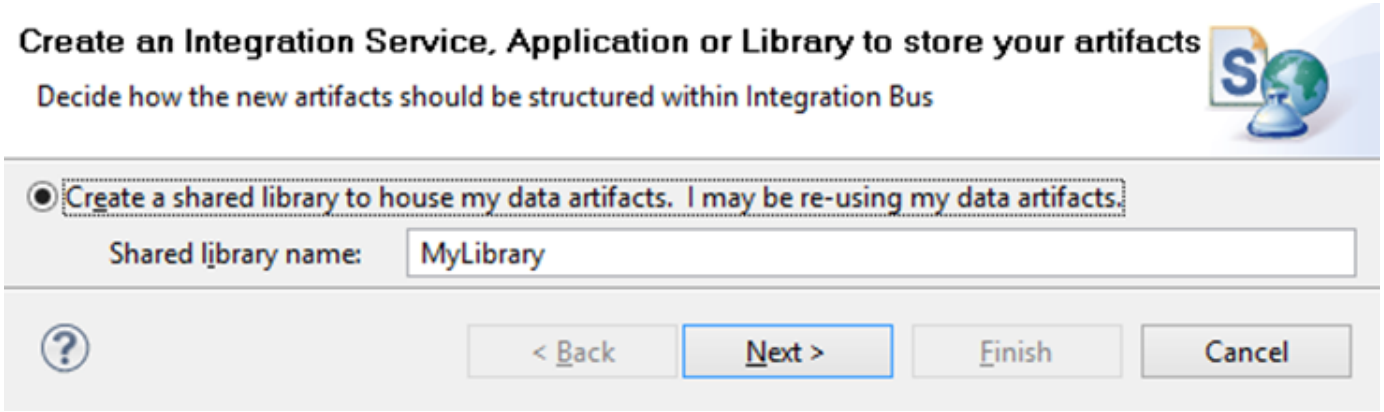
Additionally, IIB version 10 has led to one additional Message Model workaround in addition to requiring all existing Workarounds…

Use Cases (12)

- Use Cases:
  - 12a) Choose an appropriate Message Model Library type (Static or Shared) based upon Deployment Strategy and Change Control Strategy i.e. not based on an IIB product limitation [IIB v10 only]
- Additional context:
  - The choice between creating a Static Library or Shared Library for housing Message Model artefacts, or any other artefacts for that matter, should be a philosophical choice based upon code / artefact management strategies
  - User's should not be forced to use one or the other for certain tasks, unless there is good reason (backed by supplied guidance notes / documentation)
    - Certainly, the justification for using Shared Libraries should not be based upon a limitation within just one of the Message Model discovery wizards alone!
  - Static Libraries must be used with IIB version 9 since the concept of Shared Libraries was introduced in IIB version 10

Issues (12)

- The “Quick Start > Start from WSDL and/or XSD files…” wizard, which overcomes a number of issues / limitations with the other “New > Message Model” wizards when working with XSDs in IIB version 9 has been unnecessarily locked-down in IIB version 10 to force users to select “Shared Library” Projects / containers when loading message definition files into a Library



  - Static Libraries – clearly work and are strategic (they are not deprecated)
    - Within IIB version 9, Static Libraries are the only Library option
    - Within IIB version 10, Static Libraries can be used if generating Message Models using other import methods (albeit we’ve explored the other issues with those import methods already)

Issues (12)

- Additional issue factors:
  - The decision as to whether to utilise Shared vs. Static Libraries across an integration platform is a strategic one and it would be nonsensical to decide based purely on the limitation of a single wizard
  - PMR support on this issue has been weak and misleading
    - But the Support group are limited as to how much they help – there seems to be fundamental Product Architecture and consistency issues here
    - PMR 71483,001,866 – IBM suggested that the decision to lock down the “Quick Start” wizard is a strategic one in order to encourage the usage of Shared Libraries
    - Clearly, the “Quick Start” wizard is one of multiple methods to setup a Library containing a Message Model so it is inconsistent to lockdown one method and not others
    - Having unnecessary limitations imposed on an area of product functionality, which is suffering so many issues, workarounds, and inconsistencies already is not helpful for users
    - RFE 89916 has been reluctantly raised

Workarounds (12)

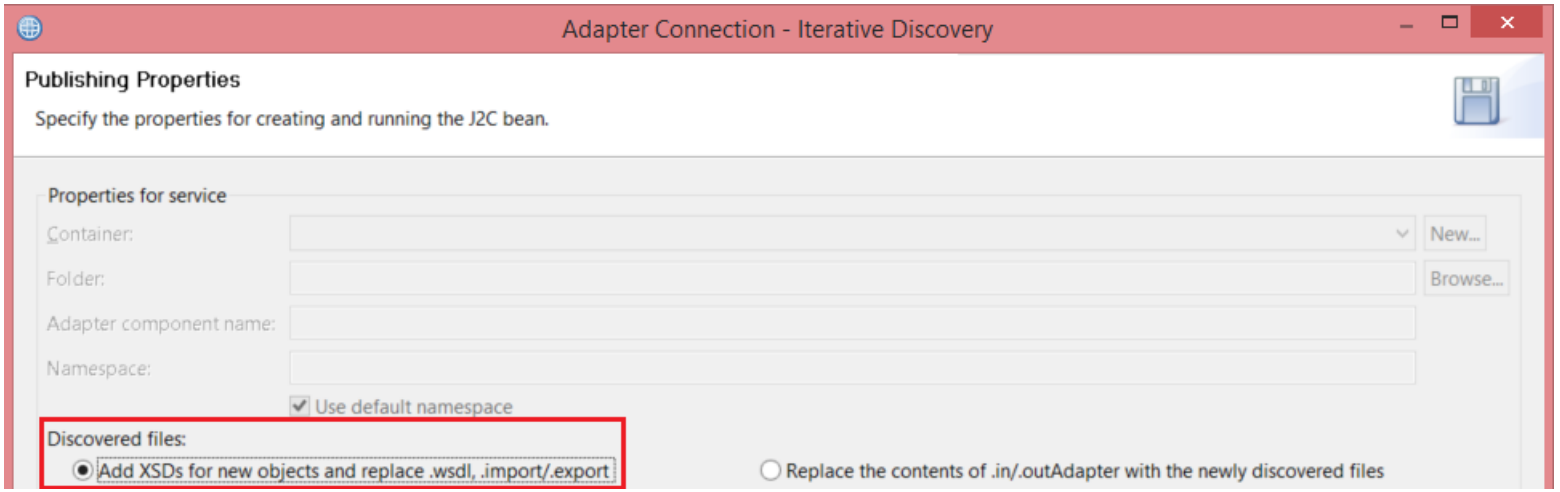
- IIB version 10 only: Use the “Quick Start” wizard to generate an **undesired Shared Library** from an XSD package
    - **Manually edit** (hack) the resultant “.project” file to convert the generated Shared Library into a Static Library
    - Refresh the Library Project within the toolkit and the respective Library Descriptor file should be refactored to reflect the change from a Shared Library to a Static Library
    - Note: Ordinarily, the “.project” file is application-maintained and users should not need to edit this file directly

Other IIB Message Model Issues?

- The following, lesser issues, are clear bugs / inconsistencies, pertinent to Message Models / working with Message Models, for which the Use Cases are simple / self-explanatory…
- A workaround for a workaround:
  - After generating a Message Model from XSDs, and applying any required Message Model workarounds e.g. defining a Message Model sub-folder within which to store the XSDs, the IIB version 10 Integration Toolkit may present an XSD warning, for example:
  - "**Global element** xxxx **in project** yyyy **exists more than once within the Library** zzzz"
  - The Integration Toolkit retains some form of memory / state of a Global Element or Global Type existing at the root of the Message Model, even after moving an XSD to a Message Model sub-folder. Cleaning the Workspace does not resolve this warning.
  - Workaround – commit / share the affected project to a source control repository for safety, delete the project from the Workspace and check-out / download the project back from the source control repository

Other IIB Message Model Issues?

- “Adapter Connection – Iterative discovery…” wizard inconsistencies
  - The Iterative Discovery wizard should allow an SAP-based Message Model to be rediscovered, with any changes (from SAP) detected and refactored into an existing Message Model seamlessly, with minimum effort involved
  - In-fact, when default Iterative Discovery options are used:



  - Inconsistencies are introduced into the updated / refactored Message Model XSDs…

Other IIB Message Model Issues?

- “Adapter Connection – Iterative discovery…” wizard inconsistencies, continued…

```
<xsd:element name="SNDPFC" nillable="true" minOccurs="0" maxOccurs="1">
<xsd:annotation xml:space="preserve">
<xsd:appinfo source="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/metadata">
<sapasi:sapALEPropertyTypeMetadata xmlns:sapasi="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/metadata">
<sapasi:FieldName>SNDPFC</sapasi:FieldName>
<sapasi:SegmentHierarchy>0</sapasi:SegmentHierarchy>
<sapasi:OffSet>0</sapasi:OffSet>
<sapasi:PrimaryKey>false</sapasi:PrimaryKey>
<sapasi:MaxLength>2</sapasi:MaxLength>
<sapasi:ForeignBOKeyRef></sapasi:ForeignBOKeyRef>
```

```
<xsd:element maxOccurs="1" minOccurs="0" name="SNDPFC" nillable="true">
<xsd:annotation xml:space="preserve">
<xsd:appinfo source="http://www.ibm.com/xmlns/prod/websphere/j2ca/sap/metadata">
<sapasi:sapALEPropertyTypeMetadata>
<sapasi:FieldName>SNDPFC</sapasi:FieldName>
<sapasi:SegmentHierarchy>0</sapasi:SegmentHierarchy>
<sapasi:OffSet>0</sapasi:OffSet>
<sapasi:PrimaryKey>false</sapasi:PrimaryKey>
<sapasi:MaxLength>2</sapasi:MaxLength>
<sapasi:ForeignBOKeyRef/>
</sapasi:sapALEPropertyTypeMetadata>
```

  - These are innocent / non-consequential differences in XSD terms:
    - Re-ordering of XSD attributes such as min / max etc
    - Duplicate / redundant namespace alias declaration
    - Change in representation of empty tags
  - Two issues here however, this inconsistent XSD styling leads to…

Other IIB Message Model Issues?

- "Adapter Connection – Iterative discovery…" wizard inconsistencies, continued…
    - Pointless changes being introduced
        - This is bad from a change control perspective, each time an alternative method of discovery / rediscovery is applied to a Message Model (iterative or non-iterative), the scope of changes made to Message Model files appears to be larger than necessary
    - Message Model contention
        - Once again – thanks to the "unified" Message Model implementation for IIB, these pointless amendments cause **widespread conflicts** to occur between SAP Message Models that are discovered / deployed **with** and **without** using the **"Iterative Discovery" wizard**
        - Every shared / common SAP XSD, including the default Control Record XSD previously discussed, conflict because their contents no longer match – once again, IBM is tripping-up over its own product limitations here
- IIB Message Models have rendered the "Iterative Discovery" wizard, **unusable**!

Compiled List of Use Cases

Compiled list of tasks which **IIB Message Models** do not support i.e. cannot be completed using Message Models / cannot be completed using Message Models without observing workarounds…

- 1a) Support multiple message definitions generated from a SAP system (i.e. for various RFC functions)
- 1b) Support multiple message definitions supplied from various third-parties
- 2a) Define a Proxy-Service that re-advertises a Service Method of a Sub-ordinate (SOAP)
- 2b) Define a Composite-Service that re-implements a Service Method of a Sub-ordinate (SOAP)
- 2c) Define a Composite-Service that interacts with one or more Sub-Ordinates (SOAP)
- 2d) Define multiple Message Flows to implement different Operations of a Service definition (SOAP)
- 3a) Split / breakdown SOAP Service definitions across / into multiple WSDL files
- 3b) Extend a WSDL file with another (SOAP) e.g. in order to specify additional endpoints
- 3c) Implement a third-party WSDL package which includes multiple WSDLs for communicating with a third-party Web Service
- 4a) Implement multiple XSD packages that contain XSD files for which no target namespace is defined
- 5a) Define an XSD-based Message Model within a Project (Application, Library etc) according to a package structure / hierarchy, following and maintaining accurate import references (XSD file dependencies)
- 6a) Use default discovery settings for generating Message Models for multiple SAP IDocs using the IBM SAP Adapter and the IBM SAP Adapter Discovery Wizard
- 7a) Use default discovery settings for generating Message Models for multiple SAP BAPIs using the IBM SAP Adapter and the IBM SAP Adapter Discovery Wizard
- 8a) Perform a patch upgrade of an IIB Integration Node / Toolkit or apply an Interim Fix to the IBM SAP Adapter for an existing Integration estate
- 9a) Deploy a change to an existing, shared XSD within an XSD package / Message Model without impacting all (other) Message Models that reference that shared XSD
- 9b) Introduce a new Message Model based upon an XSD package containing a shared XSD without impacting existing Message Models that reference that shared XSD
- 10a) Utilise (any) Source Control Keyword Substitution within message definition files (WSDLs, XSDs)
- 10b) Work with SOAP version 1.2 Message Models when using Subversion (SVN) for Source Control
- 11a) Implement two different SAP Release Versions of the same SAP IDoc
- 12a) Choose an appropriate Message Model Library type (Static or Shared) based upon Deployment Strategy and Change Control Strategy i.e. not based on an IIB product limitation [IIB v10 only]
- other) Use the IIB “Adapter Connection – Iterative discovery…” wizard

Compiled List of Issues and Workarounds

Issue: Message Definition File Contention (theme 1)

- Workaround: Define a Message Model Sub-folder

Issue: IBM Schema Extension, File Conflicts (theme 2)

- Workaround: Manually Edit IBM Schema Extensions

Issue: WSDL Filename Contention (theme 3)

- No Message Model Workaround available: Do Not Import, refactor already valid WSDLs, or alternatively, Use Message Sets

Issue: XSDs without a Target Namespace (theme 4)

- No Message Model Workaround available: Use Message Sets

Issue: Generating Message Models from XSDs [multiple issues here] (theme 5)

- Workaround: Use the Quick Start Wizard (limitations apply)

Compiled List of Issues and Workarounds

Issue: Complete file-system hierarchy not preserved for XSDs (also theme 5)

- No Message Model Workaround available: Manually construct the required filesystem structure

Issue: SAP IDoc discovery using IBM SAP Adapter causes conflicts (theme 6)

- Additional Workaround 1: Business Object Namespace Customisation
- Additional Workaround 2: Cleanse the SAP Resource Adapter Version

Issue: SAP BAPI discovery using IBM SAP Adapter causes conflicts (theme 7)

- Additional Workaround 1: Business Object Namespace Customisation

Issue: IIB Fix-pack upgrades and IBM SAP Adapter patching causes conflicts (theme 8)

- Workaround: Hack future Message Model files to cleanse or rediscover all

Issue: XSD change management (theme 9)

- Workaround: Change deployment topology, design, or Change Management approach

Compiled List of Issues and Workarounds

Issue: Source Control, Keyword Substitution causes conflicts (theme 10)

- No Message Model Workaround available: Use Message Sets

Issue: IBM SAP Adapter IDoc Release Version conflicts (theme 11)

- No Message Model Workaround available: Re-solution or use Message Sets

Issue: IIB v10 Quick Start wizard supports Shared Libraries only (theme 12)

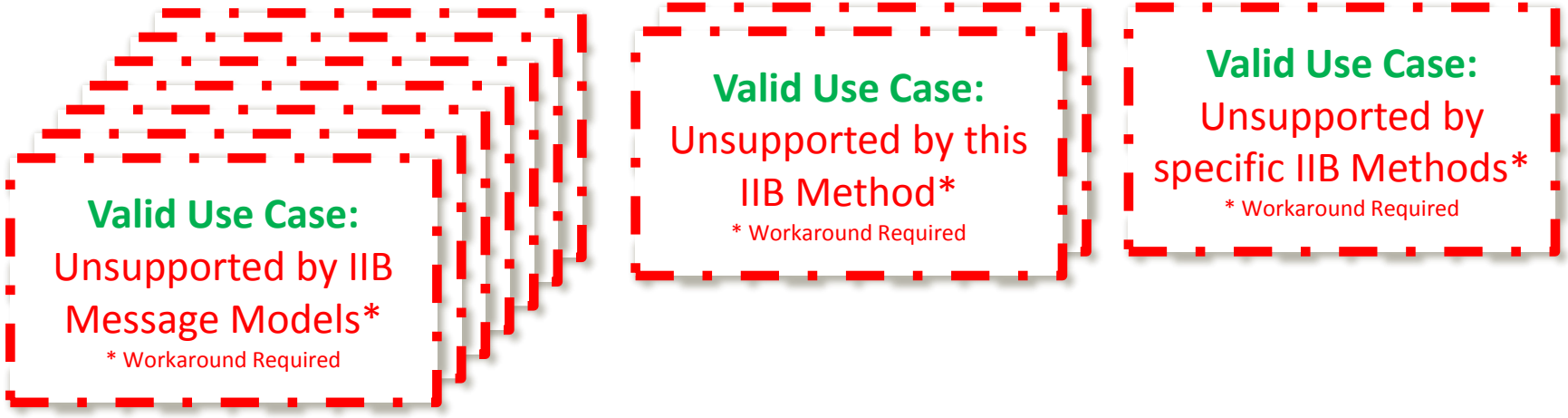
- Workaround: Hack the Library meta-data files to transform to Static Library

Issue: Iterative Discovery inconsistencies (theme “other”)

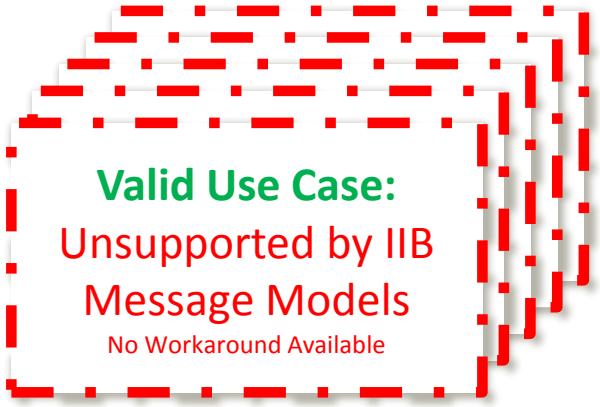
- Workaround: Don't use the Iterative Discovery wizard

Workarounds Summary

- **Ten issues** for which **IIB Message Model Workaround** is required and is identified:



- **Five issues** for which an **IIB Message Model based Workaround** is not available – the problem must be resolved elsewhere / outside of the product set / or the valid Use Case avoided:



Root Cause Analysis – Issue Categorisation

Root Cause: The "unified" Message Model / lack of Message Model isolation between models

- **Issue**: Message Definition File Contention
- **Issue**: IBM Schema Extension, File Conflicts
- **Issue**: XSDs without a Target Namespace
- **Issue**: SAP IDoc discovery using IBM SAP Adapter causes conflicts
- **Issue**: SAP BAPI discovery using IBM SAP Adapter causes conflicts
- **Issue**: IIB Fix-pack upgrades and IBM SAP Adapter patching causes conflicts
- **Issue**: XSD change management
- **Issue:** Source Control, Keyword Substitution causes conflicts
- **Issue**: IBM SAP Adapter IDoc Release Version conflicts

Root Cause: Backwards compatibility issue caused by change in Message Model generation approach

- **Issue**: WSDL Filename Contention
- **Issue**: Complete file-system hierarchy not preserved for XSDs

Root Cause: Inconsistent implementation – flawed approach or specific bug

- **Issue**: Generating Message Models from XSDs [multiple issues here]
- **Issue**: IIB v10 Quick Start wizard supports Shared Libraries only
- **Issue**: Iterative Discovery inconsistencies

Root Cause Analysis – Issue Categorisation

Root Cause: The "unified" Message Model / lack of Message Model isolation between models

- **Issue**: Message Definition File Contention
- **Issue**: IBM Schema Extension, File Conflicts
- **Issue**: XSDs without a Target Namespace
- **Issue**: SAP IDoc discovery using IBM SAP Adapter causes conflicts
- **Issue**: SAP BAPI discovery using IBM SAP Adapter causes conflicts
- **Issue**: IIB Fix-pack upgrades and IBM SAP Adapter patching causes conflicts
- **Issue**: XSD change management
- **Issue:** Source Control, Keyword Substitution causes conflicts
- **Issue**: IBM SAP Adapter IDoc Release Version conflicts

Root Cause: Backwards compatibility issue caused by change in Message Model generation approach

- **Issue**: WSDL Filename Contention
- **Issue**: Complete file-system hierarchy not preserved for XSDs

Root Cause: Inconsistent implementation – flawed approach or specific bug

- **Issue**: Generating Message Models from XSDs [multiple issues here]
- **Issue**: IIB v10 Quick Start wizard supports Shared Libraries only
- **Issue**: Iterative Discovery inconsistencies

…The "unified" Message Model exacerbates the impact of these other issues

Root Cause Analysis – Issue Categorisation

Root Cause: The "unified" Message Model / lack of Message Model isolation between models

- **Issue**: Message Definition File Contention
- **Issue**: IBM Schema Extension, File Conflicts
- **Issue**: XSDs without a Target Namespace
- **Issue**: SAP IDoc discovery using IBM SAP Adapter causes conflicts
- **Issue**: SAP BAPI discovery using IBM SAP Adapter causes conflicts
- **Issue**: IIB Fix-pack upgrades and IBM SAP Adapter patching causes conflicts
- **Issue**: XSD change management
- **Issue:** Source Control, Keyword Substitution causes conflicts
- **Issue**: IBM SAP Adapter IDoc Release Version conflicts

Root Cause: Backwards compatibility issue caused by change in Message Model generation approach

- **Issue**: WSDL Filename Contention
- **Issue**: Complete file-system hierarchy not preserved for XSDs

Root Cause: Inconsistent implementation – flawed approach or specific bug

- **Issue**: Generating Message Models from XSDs [multiple issues here]
- **Issue**: IIB v10 Quick Start wizard supports Shared Libraries only
- **Issue**: Iterative Discovery inconsistencies

…This is the key product issue, with zero Customer benefits for us

Root Cause Analysis – Critique

Root Cause: The "unified" Message Model / lack of Message Model isolation between models

- This can only be described as a flawed product architecture from a Customer perspective
- Message Sets were ugly / fiddly to work with, but their implementation was robust. Message Models may offer benefits but without the equivalent robustness that Message Sets has to offer
- There have been Customer consequences to the lack of Message Model isolation, as well as IBM development consequences
- IBM has not plugged the gaps within its own product where fallout because of this change in Message Modelling architecture has occurred

Root Cause: Backwards compatibility issue caused by change in Message Model generation approach

- Bad approach to **backwards-compatibility**
- From a Support perspective, we are told that fixing these issues would cause a backwards-compatibility issue
- These issues are primarily backwards-compatibility issues for Message Broker users migrating to IIB, secondary to that, they are issues in their own right with regards to IIB itself
- Fixing a backwards-compatibility only causes backwards-compatibility problems when they are not fixed in a timely manner
- Offering configuration "options" would allow concerns to be addressed without impacting users who wish to continue using current functionality (a more flexible approach)

Root Cause: Inconsistent implementation – flawed approach or specific bug

- "working as designed" isn't always a reasonable excuse – whilst there is complete **inconsistency** with the end result obtained, when simply choosing between two **alternative access methods**, within the **same toolkit**, to complete the **same task** (supposedly)
- Either the design is wrong or there is a product bug
- Architecturally why isn't the same function being used in the background to complete the task, irrespective of how the wizard was launched?
- If it's not clear to seasoned users, how will it appear to new users of the product?

Solution Analysis - Options

Root Cause: The "unified" Message Model / lack of Message Model isolation between models

- **Stop** the pooling all Message Models together into a **"unified" model**, expecting compatibility across unrelated Message Models i.e. re-introduce the same levels of isolation that existed with Message Sets
- Additionally/optionally: **Enable** the same **Message Model / message type selection** that was available with Message Sets – it's quite acceptable for an integration developer to need to choose / understand what message they expect / are dealing with at any one time – else what are they validating against, or coding towards?

Root Cause: Backwards compatibility issue caused by change in Message Model generation approach

- Provide more flexibility when handling of both WSDL/XSD packages and XSD packages…
- Allow users to choose their desired Message Model generation method i.e. drive package path generation from: **target namespace** (as per Message Broker legacy), or source **filesystem structure** (as per IIB) but this latter option need one further customisation…
- With / without "**preserve relative directories only**" option

Root Cause: Inconsistent implementation – flawed approach or specific bug

- Accept that illogical, undocumented inconsistencies are issues / bugs and fix them…
- Where there are multiple ways to access a Message Model wizard for generating a Message Model from a specific source – all methods should achieve the same result, all methods should offer the same flexibility in terms of options

Solution Analysis - Summary

- A **single change** in product architecture would **solve ALL major** Message Model **issues** – i.e. solving the "unified" Message Model issue by introducing isolation between Message Models
    - Ironically the change proposed would take Message Models closer to how their predecessor (Message Sets) worked, in other words, it's a proven approach
        - …therefore, a less risky proposal than the proposal to introduce "unified" Message Models in the first place
    - Additionally, let users specify a Message Model explicitly at runtime (as with Message Sets) – being explicit is more efficient, more purposeful – more informative (it clarifies intention)…
        - Have we really stream-lined the development process with the current product functionality?
        - Explicitly Message Model / message type selection is not a mandatory co-requisite but it certainly could help…
            - Ensures there is no ambiguity if Message Model isolation is introduced (the same global element may appear within multiple Message Models), no need to create a "unified" Message Model in order to de-duplicate, or pick-the-first-match (circumstantial)

Solution Analysis - Summary

- The remaining issues are not **as severe**, but carry multiple, negative consequences
    - They represent a mixture of regressed functionality (in terms of the transition from Message Broker to IIB), and inconsistencies
    - It's the total number of issues, and workarounds that have to be observed which make the current product very difficult to work with
    - Some of these issues should be relatively easy to solve, but there appears to be a lack of appetite to do so
- Smaller stumbling blocks present challenges when attempting to derive workarounds for larger issues, they narrow the number of alternative approaches available
    - Navigating a large number of workarounds is difficult enough without having options further reduced
- Left as-is, the list of issues may still grow
- Once again, these Use Cases can be **wholly fulfilled**, and **without workaround**(s) when implementing Message Sets… The current Message Model implementation is hardly an evolutionary step-forward in this respect

Solution Analysis - Summary

- Even if a major, one-fix solution for a set of issues is not undertaken, smaller, individual solutions would improve the user experience for / or resolve specific issues
    - E.g. issue (11) is resolvable with an IBM fix to label all IBM-generated SAP Message Model files using the IDoc Version tag (to differentiate between otherwise conflicting versions)
        - This would stop one IBM IIB component (the IBM SAP Adapter) from causing contention with another IBM IIB component (IIB Message Models, within an Application or Library)
        - This is the type of impact analysis that should have been undertaken when evolving the way that the IBM SAP Adapter is used with IIB
            - The contention that is occurring between multiple parts of the same shippable product, is not Customer driven in some cases – somewhat embarrassing on the part of IBM
            - So why not fix it? Is the product really working as designed, or failing to entirely work because one part of its design has evolved whilst the other has been left lagging?
        - IBM Support Case TS000117159 has been raised for this issue
    - This is just one example where a local fix could resolve a depicted issue, there are others, but to-date the IBM Support process has not been helpful with regards to Message Model contention issues
        - Our general observation is that local fix, after local fix, would be a sign that there's something more fundamental going on here – each local fix may simply be fighting against a bigger issue

Unified / Pooled Message Models

- Pooling multiple Message Models together to form a “unified” Message Model, fully searchable by any code unit within scope (e.g. IIB Application-scope, or IIB Integration Server-scope in the case of Independent Resources) is in many respects “restrictive”…
- There is an assumption that pooled models will be compatible with one another
  - What if they are not? What should be done in these scenarios?
  - Models may come from many sources (in-house defined, supplied by third parties, acquired from standards bodies / frameworks etc)
    - The world is not strictly organised, so why should a heterogeneous set of message definition packages be assumed to be compatible with one another?
- IBM have not managed to deal with the impact of this pooling / unification within the IIB product itself, the impacts even render some existing (not deprecated) product feature unusable e.g. the “Adapter Connection – Iterative discovery…” wizard inconsistencies
- IBM with IIB has lost sight of what integration is partly here for…
  - Integration is here to deal with inconsistencies in other applications / interfaces / APIs
  - It is one of the bread-and-butter functions that integration platforms / teams perform

Unified / Pooled Message Models

- If two sets of message definitions, representing two interfaces / two sides of a mediation flow are not compatible with one another…
    - That might be the principal reason why integration is required in the first place, to mediate between the two incompatible application interfaces
    - The current IIB Message Model architecture would require us to down-tools under such circumstances
        - …If applying a myriad of workarounds does not suffice!
- For someone new to the IIB product, these issues / restrictions may seem bizarre, for someone experienced with the its predecessor (Message Broker / Message Sets), this “evolution” is disappointing
- IBM’s own life has been made more difficult because of IIB Message Models – any future product fix / upgrade / enhancement that results in any change whatsoever to the content of a generated Message Model XSD (even a non-consequential change) will immediately cause a backwards compatibility issue
    - As has been the case with the “Adapter Connection – Iterative discovery…” wizard inconsistencies
    - These issues don’t appear to be understood / accepted IBM, therefore the pain of dealing with such backwards compatibility issues will be passed onto Customers… to derive further workarounds?

Unified / Pooled Message Models

- We've seen examples of how "unified" Message Models inhibit granular change management
  - Potentially forcing one Message Model to change because of another
    - A cardinal sin in integration terms to force change on a related, let alone unrelated interface!
  - And / or, potentially requiring a complex rollout strategy (delete + deploy rather than redeploy)
  - Again, inhibiting, inflexible – these aren't aspirational qualities for a product upgrade
- There is a lack of clear procedural guidance from an IIB perspective – how to avoid contention issues
- Also, lack of transparency in terms of "What's new?", "What's changed?" / their consequences
- If this documentation is added to a "What's new?" article, some existing Customers may choose to avoid upgrade, or continue using older product functionality
  - Nonetheless it is an honest account of what has changed and would save Customers' time
  - The isolation levels for Message Models have a severe impact on how message definitions need to be managed
  - And still some issues are unresolvable within the product itself / no product workaround is available

Unified / Pooled Message Models

- In terms of the IBM Support process
  - In some cases, Customers are being asked to raise “enhancement” requests to get back the functionality they had in the first place
  - Other feedback received includes that issues cannot be fixed due to “backwards compatibility concerns”
    - When from a Customer-perspective, the issue being reported itself is a backwards compatibility issue – not the fix
    - And, there are solution options that would allow users to access both **current** and **corrected** behaviours (e.g. exposing a configuration option / radio selection button within a wizard)
- The multitude of Message Modelling issues leads to a growing perception that they are not fit-for-purpose, not a viable replacement for Message Sets in many scenarios
  - DFDL has not been explored within the scope of this presentation
    - DFDL itself may highlight advantages in the strategic use of Message Models (as opposed to MRM Message Sets) for modelling bespoke message formats, including usability improvements
  - Core issues highlighted within this presentation are focused on the “mechanics” of how Message Models operate, irrespective of message format / domain - issues with regards to scope / levels of isolation etc. The principles of contention behind issues are universally applicable

IBM Support References

Support Ticket Reference	Issue Description	IBM Support Response	APAR	APAR Problem Description	RFE	RFE Status
PMR 45598,001,866	"New Message Model" Wizard, XSD hierarchy issue	Initially refused to acknowledge that any problem exists, then provided several approaches to dealing with this issue which do not work Finally, the core issue reported has not been fixed (RFE requested) A lesser issue identified has been fixed (see APAR)	IT05625	The New Message Model wizard does not allow an existing XSD file to be overwritten.	63935	Uncommitted Candidate Last updated: 07 Oct 2015
PMR 45919,001,866	When trying to import XSDs into a Message Model, the toolkit is "sometimes" adding 'ibmSchExtn' prefixes to the global elements, sometimes it's not, causing Message Model conflicts	Workaround suggested by IBM support - manually add the missing declarations for all global elements, preventing contention if the same Messaging XSD appears in multiple message models that are referenced from the same application. APAR IT05373 raised for IBM to resolve this issue. IBM later stated that they no longer wish to fix the issue, the toolkit team can apply the fix to ignore these differences but the XSD discrepancy would still cause an issue with the runtime. Finally the APAR was re-instated with a revised solution (see APAR). However the original issue has not been fully resolved e.g. for Independent Resources, the toolkit won't "highlight" contention and so no quick-fix will be supplied, contention can occur at deploy-time. Message Model files need to be manually edited (existing / deployed Message Models may need to be reworked - regression impact)	IT05373	When trying to import XSDs into a Message Model, the toolkit will add 'ibmSchExtn' prefixes to the global elements depending on what is selected in the import. When multiple message models are linked to the same application this can result in an error. The toolkit now offers a quick fix option for any schemas that are highlighted as a problem. If the problem occurs, click on the error message and select quick fix. This will display a wizard that allows the additional attribute to be removed.	-	-

IBM Support References

Support Ticket Reference	Issue Description	IBM Support Response	APAR	APAR Problem Description	RFE	RFE Status
PMR 46338,001,866	Message Model - WSDL vs. XSD Import inconsistencies WSDL hierarchy is not preserved (XSD hierarchy is) whilst generating a Message Model from a WSDL	This is due to an undocumented change in behaviour in the product: "the old MSET scheme, which used folder name based on namespace, did not have this issue. But the problem with that scheme was folder paths names were too long and we have to either filter or map characters in namespace URI which are not valid for file path etc.. Some customers did not like it and they wanted to see wsdls into the folder they selected for import." Two workarounds suggested by IBM but neither of these are correct / successful. Finally, the core issue reported has not been fixed (RFE requested)	IT08094	The WSDL importer did not handle re-importing resources correctly when the wsdl and schema files already existed in the target location.	71301	Submitted Last updated: 14 Jan 2016
			IT09062	Documentation update: If a WSDL, that includes/imports multiple other WSDL files that have the same or different namespaces, is imported into an application or library, the user will notice in the application or library root folder (or specific folder provided by the user on the WSDL import page), WSDL files of same name with monotonically increasing number appended to them. This is not described in the product documentation.		
PMR 71483,001,866	Message Model - XSD generation restriction Quick Start Wizard forces you to generate Message Models into a Shared Library when working with XSDs (does not allow Static Libraries to be created)	IBM suggested that the decision to lock down the Quick Start wizard is a strategic one to encourage the usage of Shared Libraries. This doesn't make sense since the Quick Start wizard is one of multiple methods to setup a Library so it's inconsistent to lock this one method down and not the others. Workaround suggested to manually hack the '.project' file after initial Message Model generation. The issue reported has not been fixed (RFE requested)	-	-	89916	Need More Information (this workshop) Last updated: 29 Jun 2016

IBM Support References

Support Ticket Reference	Issue Description	IBM Support Response	APAR	APAR Problem Description	RFE	RFE Status
PMR 72799,001,866 / Case TS000038595	IBM Defined XSDs Message Model contention - SOAP 1.2 XSD SVN keyword appears within the IBM defined SOAP 1.2 Message Model XSD causing contention to occur when these XSDs differ (due to SVN keyword substitution)	A change has been proposed by IBM to allow the IBM Defined XSDs to be shipped with the product at runtime, resolving the contention issue with their XSDs. RFE 96680 raised to cover this change, as requested by IBM via the PMR / on a Customer call. Fixes supplied but not successful - discarded. Awaiting IBM to investigate the cause of the remaining issues and rebuild the fix (with IBM since December 2016)	TFP72799	Not available / cannot be located online	96680	Submitted Last updated: 15 Jan 2017
<Raised directly as RFE>	Message Model XSD hierarchy observes the original file-system path, used to observe namespace hierarchy In the case of URNs there is absolutely no reason why Customer XSDs and/or Message Model XSDs cannot be stored using a file-system structure that is consistent with the namespace definition itself - mimicking Java Package-styles obtained when generating class objects from a set of XSDs. Options could be provided, where feasible, to provide Customer's with flexibility and allow them to maintain previous behaviour(s) - for backwards compatibility / consistency purposes.	See RFE comments - long exchange Main concerns not addressed e.g. IBM comment "Regarding the point about how we treat namespaces defined using URNs, I would be concerned not to introduce different behaviour in terms of the final file system location for schemas whose targetNamespaces are defined using URNs as oppose to behaviour where targetNamespaces are defined using URIs" We have not requested an inconsistency depending on the style of namespace schema employed (URI vs. URN), just that Customers could be given a choice "generate based on filesystem", "generate based on namespace" (whatever namespace scheme is used) allowing Message Broker users migrating to IIB to maintain like-for-like functionality.	-	-	71726	Declined Last updated: 06 Jul 2016

IBM Support References

Support Ticket Reference	Issue Description	IBM Support Response	APAR	APAR Problem Description	RFE	RFE Status
Case TS000117159	Generated SAP Message Model Contention for multiple IDoc Versions	TBC, in-progress				

Thank You For Your Time

End