IBM z Systems

TPFUG – DFDL Education

Bradd Kadlecik, TPF Development Lab

Mar 25,2015

Disclaimer

- Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

Agenda

- DFDL APIs
- XML schema – defining the logical format
- z/TPF supported DFDL annotations – defining the native format
- DFDL expressions

DFDL APIs

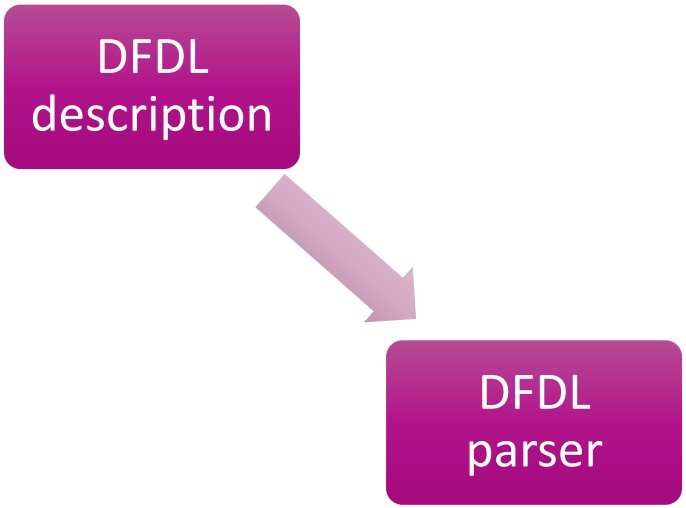
Creating XML from DFDL

```
#include <tpf/cdfdl.h>      /* DFDL APIs */
#include <tpf/c_node.h>     /* tpf_doc APIs */
#include <exception>        /* DFDL error return */

XMLHandle xmlHandle;        /* handle for the XML parser */
DFDLHandle dh;              /* handle for DFDL parser */
char *responseXML;          /* XML document output */
int rc = 0;                 /* rc for tpf_doc APIs */
struct mydata buf;          /* data to process to XML */
int  messageLen = 0;        /* XML document size */

#define DFDL_FILE "mytest.dfdl.xsd"
#define DFDL_ROOT "myStruct"
```

Creating XML from DFDL (cont'd)

```
rc = tpf_doc_initialize_handle(&xmlHandle, B2B_XML_SCANNER, NULL);
if (rc != 0) {    // error   }
rc = tpf_doc_createXMLstructure(xmlHandle,  "1.0", TPF_CCSID_UTF8,
                          TPF_CSNAME_UTF8, STANDALONE_NOT_USED,0);
if (rc != 0) {    // error   }
try {
  tpf_dfdl_initialize_handle(&dh, DFDL_FILE, DFDL_ROOT, 0);
  tpf_dfdl_setData(dh, &buf, sizeof(buf));
  tpf_dfdl_parseData(dh, xmlHandle, NULL, 0);
  responseXML = tpf_doc_buildDocument (xmlHandle, &messageLen, 0);
}
catch (std::exception & exception) {      // error       }
tpf_dfdl_terminate_handle(dh);
```

Identify the DFDL description



Identify the DFDL description

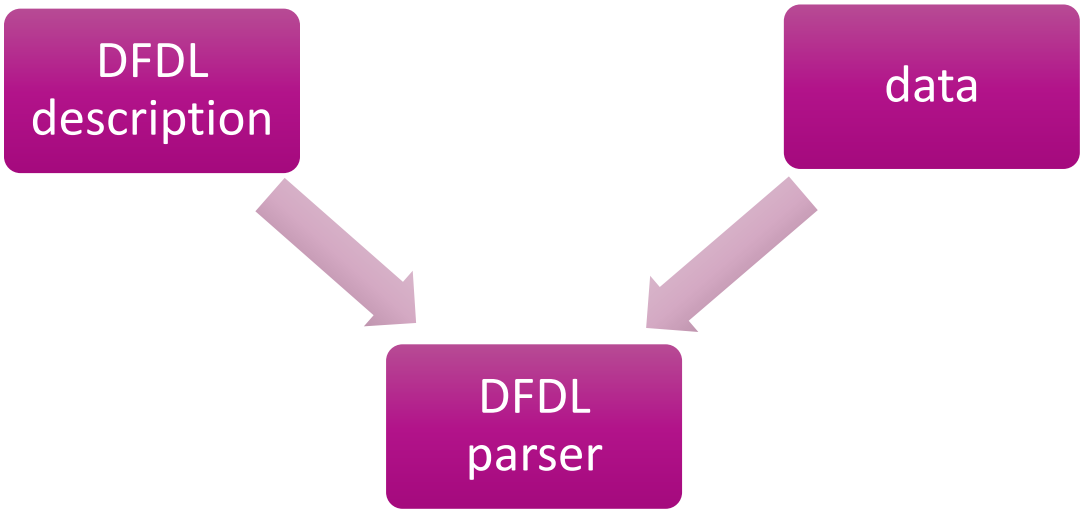
```
tpf_dfdl_initialize_handle(&dh, DFDL_FILE, DFDL_ROOT, 0);
```

1. dfdlhdl - A pointer to the location to store the DFDL API handle that is associated with the DFDL structure to be processed. This function initializes the handle and allocates the required ECB heap storage for the handle.
2. schema_file - A pointer to the name of the DFDL schema file that was loaded to the /sys/tpf_pbfiles/tpf-fdes directory by using common deployment.
3. root_element - A pointer to the name of a complex element within the DFDL schema file that defines the binary data to be processed. The root element denotes the starting and ending point of the binary data.
4. options - Specifies the processing options for this function. Specify one of the following values:

   TPF_DFDL_VERIFY - Verify that the external references in the DFDL schema file are valid. Specifying this parameter ensures that all DFDL schema files that compose the data description are loaded to the z/TPF system by using common deployment. This parameter also ensures that the referenced complex type names exist.

   0 - No special processing is requested.

Identify the data to process

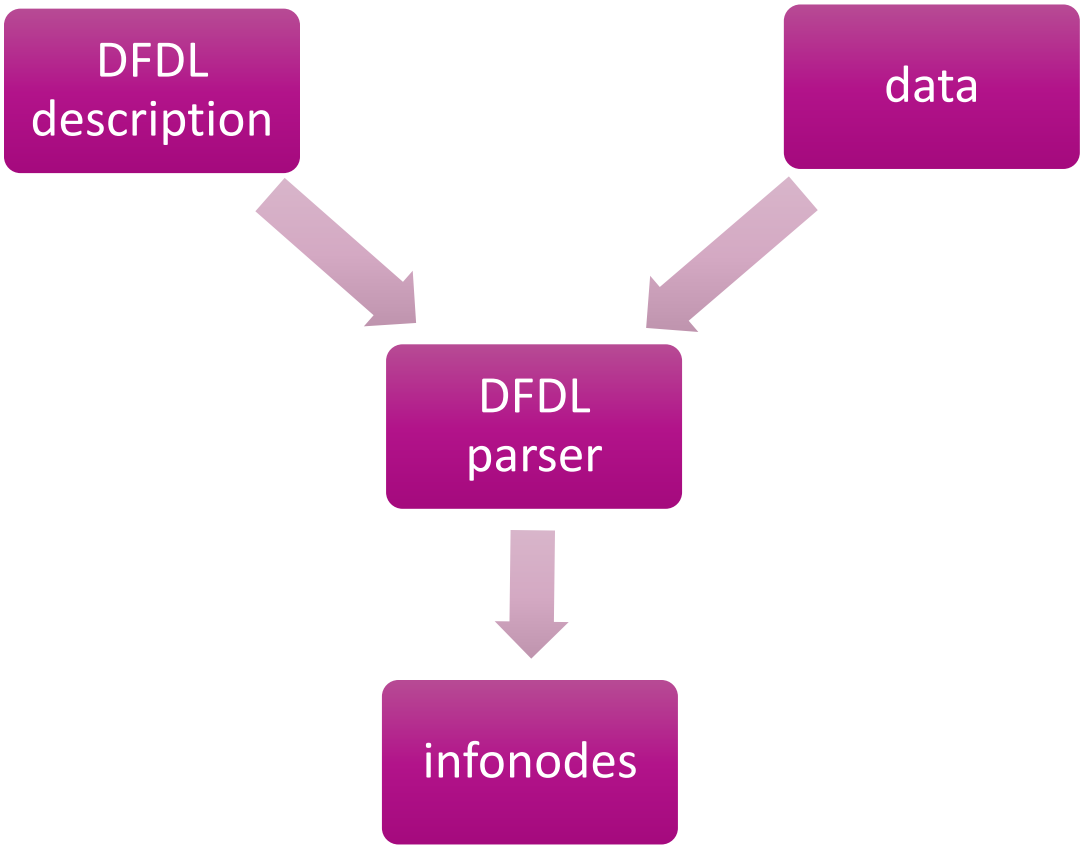


Identify the data to process

```
tpf_dfdl_setData(dh, &buf, sizeof(buf));
```

1. dfdlhdl - A DFDL API handle associated with the DFDL structure that describes the binary data.
2. buffer - A pointer to a buffer area that contains the address of a data stream that is described by a DFDL schema.
3. length - The length of the data stream that is provided by the buffer parameter. This parameter specifies the maximum length of the binary data. This value is used to ensure that computed element offsets and data accesses (for computed offsets) are within the buffer area; otherwise, a std::length_error exception is issued. If the value is 0, this validation is not performed.

Create the infonode structures



Create the infonode structures

```
tpf_dfdl_parseData(dh, xmlHandle, NULL, 0);
```

1. dfdlhdl - A DFDL API handle that is associated with a data stream and a DFDL structure to be processed.
2. api_handle - An XML or JSON API handle associated with the document structure that is being processed. The XML or JSON structures are created for the data stream as a child of the element that is specified by the doc_ElementTagName parameter. The structures are created based on the current position of the API handle.
3. doc_ElementTagName - A pointer to the name of the XML or JSON element that will be used as the parent to items that are added by the API to the XML or JSON structure. If the root of the XML or JSON structure is required, specify NULL for this parameter.
4. option - Specifies the processing options for this function. Specify the following value:

   0 - No special processing is requested.

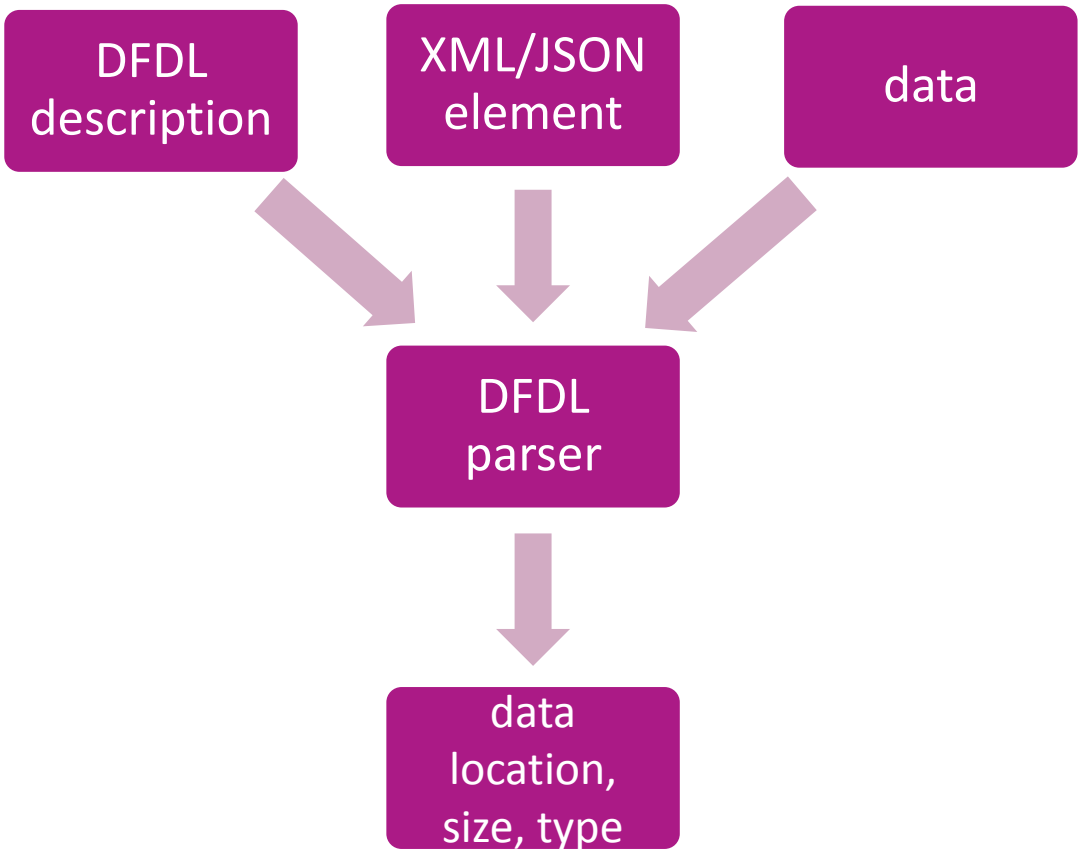
Cleanup

```
tpf_dfdl_terminate_handle(dh);
```

1. dfdlhdl - A DFDL API handle that is associated with a DFDL structure.

This API cannot throw an exception.

Accessing data by logical format



Accessing data by logical format

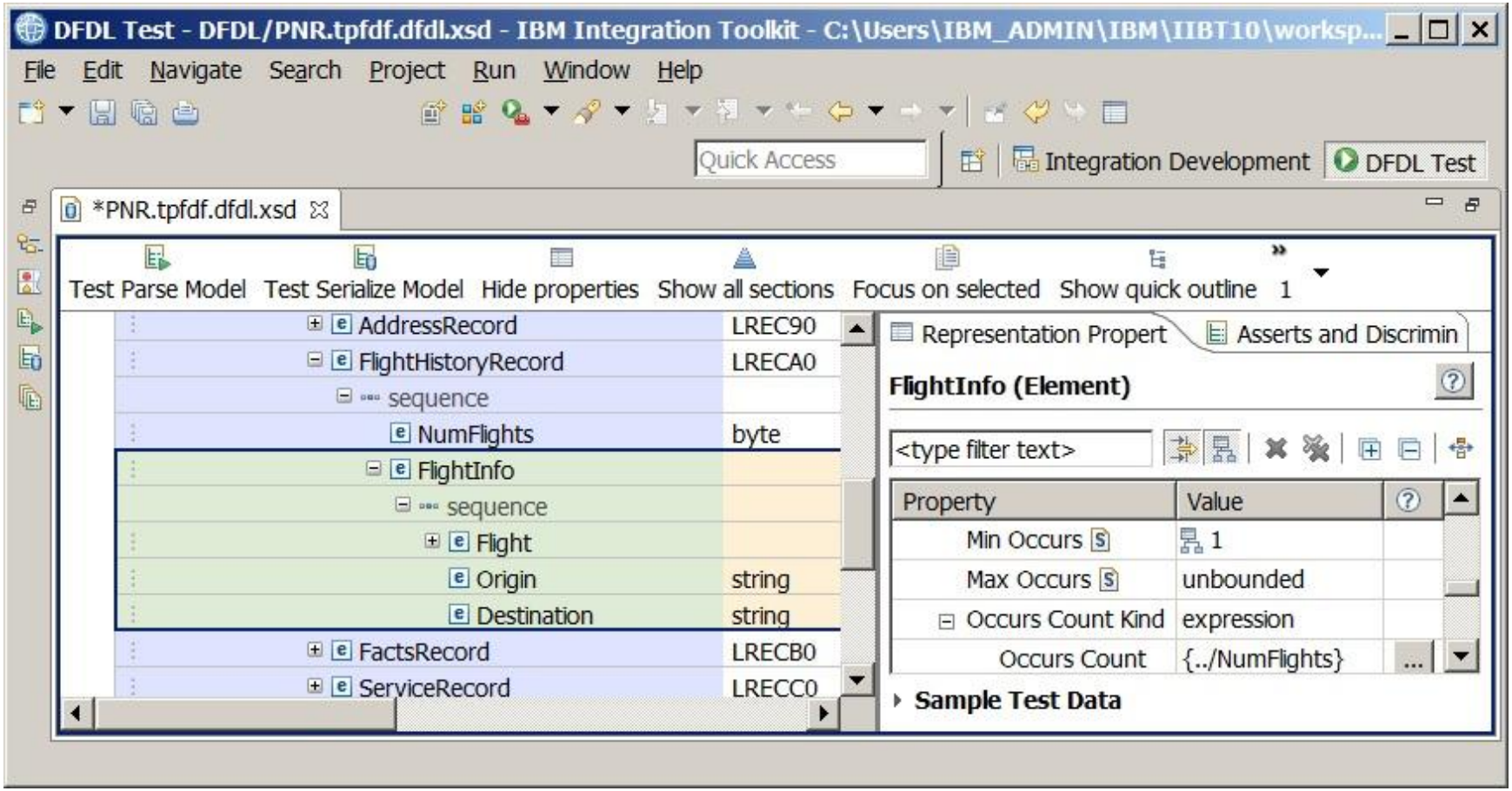
```
dfdlNodeInfo *pElementInfo;
char *pgmName;
pElementInfo = tpf_dfdl_getElementInfo(dh,"ProgramName",NULL,1,0);
pgmName = buf + pElementInfo->offset.byteInfo.bytes;
```

1. dfdlhdl - A DFDL API handle that is associated with the DFDL structure to be processed.
2. element_name - A pointer to the name of the element in the data stream that you want information about.
3. xml_namespace - A pointer to the XML namespace of the element. If you do not specify this parameter, the namespace is the same as the namespace of the schema file that is used to create the DFDL API handle.
4. occurrence - The occurrence of the element in the data stream when the element is an array. A value of 0 is not valid.
5. options - Specifies the processing options for this function. Specify one of the following:

   TPF_DFDL_VERIFY - Verifies that the element is valid for the data stream. The DFDL discriminators are used to validate the paths in the parse tree.

   0 – Requests no special processing.

Accessing Destination within FlightInfo



Accessing elements within arrays

```
DFDLHandle ArrayHdl;
ArrayHdl = tpf_dfdl_createChildHandle(dh, "FlightInfo", NULL, 3,
0);
pDestInfo = tpf_dfdl_getElementInfo(ArrayHdl, "Destination", NULL,
1, 0);
```

1. dfdlhdl - The parent DFDL API handle that is associated with the DFDL structure to be processed.
2. element_name - A pointer to the name of a complex element in the data stream that you want to create the new handle for.
3. xml_namespace - A pointer to the XML namespace of the element. If you do not specify this parameter, the namespace is the same as the namespace of the schema file that is used to create the parent DFDL API handle.
4. occurrence - The occurrence of the complex element in the data stream when the element is an array. A value of 0 is not valid.
5. option - Specifies the processing option for this function. Specify the following value:

   0 - No special processing is requested.

XML Schema

XML Schema components

- schema definition
- import/include declarations
- complexType definitions
- root element definitions

Schema Definition

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/
tpfdf/PNR"
    xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/"
    xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions"
    xmlns:ns0="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/PNR"
    xmlns:tddt="http://www.ibm.com/xmlns/prod/ztpf/xml/lib/
tpfattributes"
    xmlns:tpfbase="http://www.ibm.com/xmlns/prod/ztpf/dfdl/lib/
tpfbase">
```

XML Namespaces

- A key component of a schema definition is defining the namespaces.
- A namespace is similar to a header file name in C in that it's a unique identifier for a resource containing definitions you want to have access to.
- The targetNamespace is the unique identifier for all schema components in the current document.

z/TPF Namespace Convention for DFDL

- z/TPF namespace:
  http://www.ibm.com/xmlns/prod/ztpf
- DFDL filenames are appended in reverse order to create a unique namespace per filename.
  - PNR.tpfdf.dfdl.xsd becomes:
    dfdl/tpfdf/PNR
  - Resultant namespace:
    http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/PNR

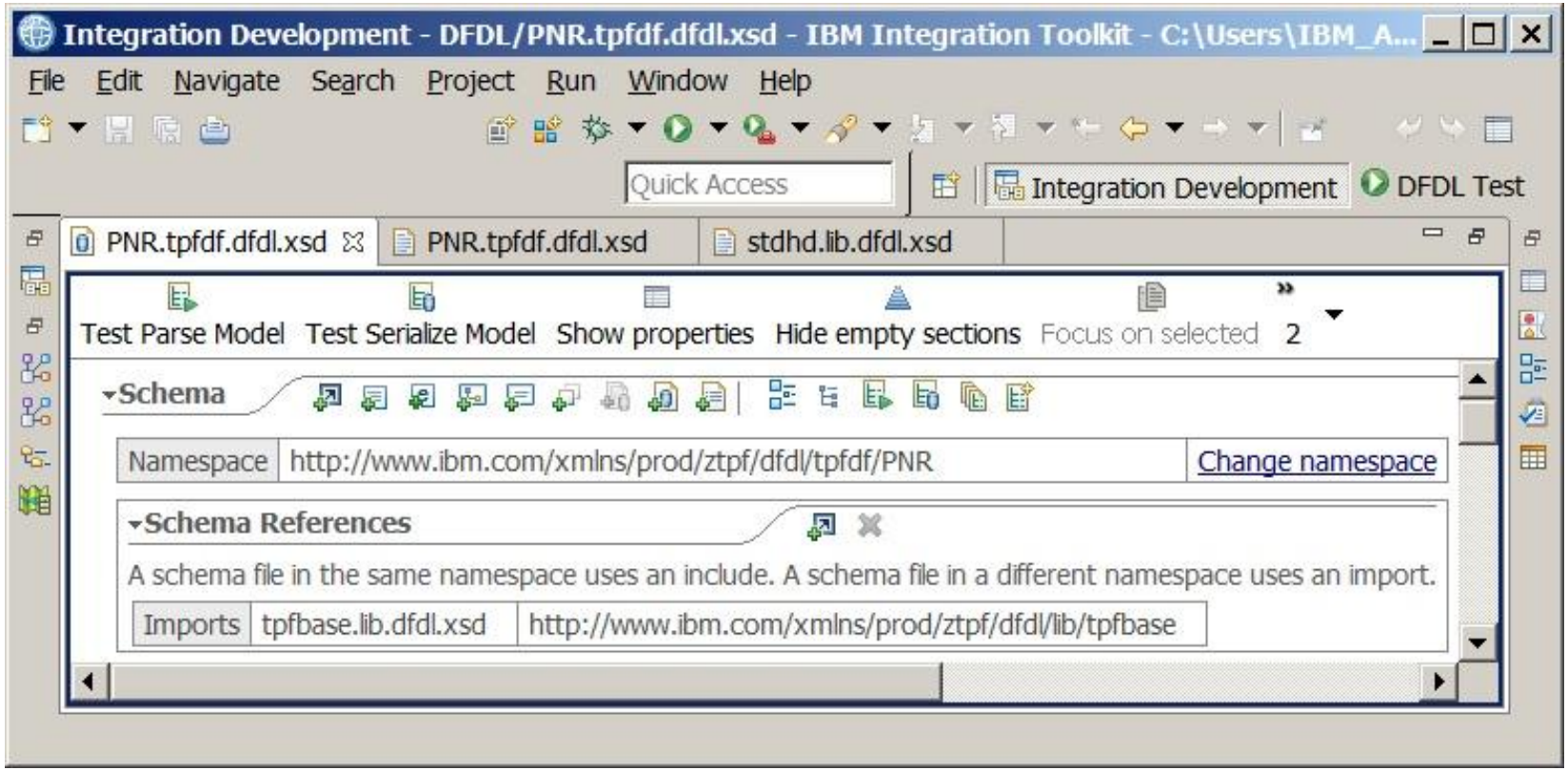
z/TPF DFDL Filename Convention

- Format: **<filename>.<resource type>.dfdl.xsd**
- DFDL filenames end in .xsd (XML schema definition) to make them easily recognizable to XML or DFDL editors.
- The file extension of .dfdl.xsd allows common deployment to recognize it as a DFDL file.
- Examples of **<resource type>:**
  - tpfdf: TPFDF definitions
  - lib: commonly referenced definitions
  - de/se: data event/signal event message
  - usr: other user defined definitions

targetNamespace for PNR.tpfdf.dfdl.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/
tpfdf/PNR"
    xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/"
    xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions"
    xmlns:ns0="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/PNR"
    xmlns:tddt="http://www.ibm.com/xmlns/prod/ztpf/xml/lib/
tpfattributes"
    xmlns:tpfbase="http://www.ibm.com/xmlns/prod/ztpf/dfdl/lib/
tpfbase">
```

targetNamespace in the DFDL editor



Namespace Prefixs

- XML uses qualified names
  - qualified name format:
    ```
    <namespace prefix>:<name>
    ```
  - Example:
    ```
    <xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    ...
    ```
- The namespace prefix helps create uniqueness when multiple XML schema/DFDL documents are involved.
- Namespace prefixs are assigned using:
  `xmlns:<namespace prefix>=<namespace>`

Namespace Prefixs for PNR.tpfdf.dfdl.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/
tpfdf/PNR"
    xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/"
    xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions"
    xmlns:ns0="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/PNR"
    xmlns:tddt="http://www.ibm.com/xmlns/prod/ztpf/xml/lib/
tpfattributes"
    xmlns:tpfbase="http://www.ibm.com/xmlns/prod/ztpf/dfdl/lib/
tpfbase">
```

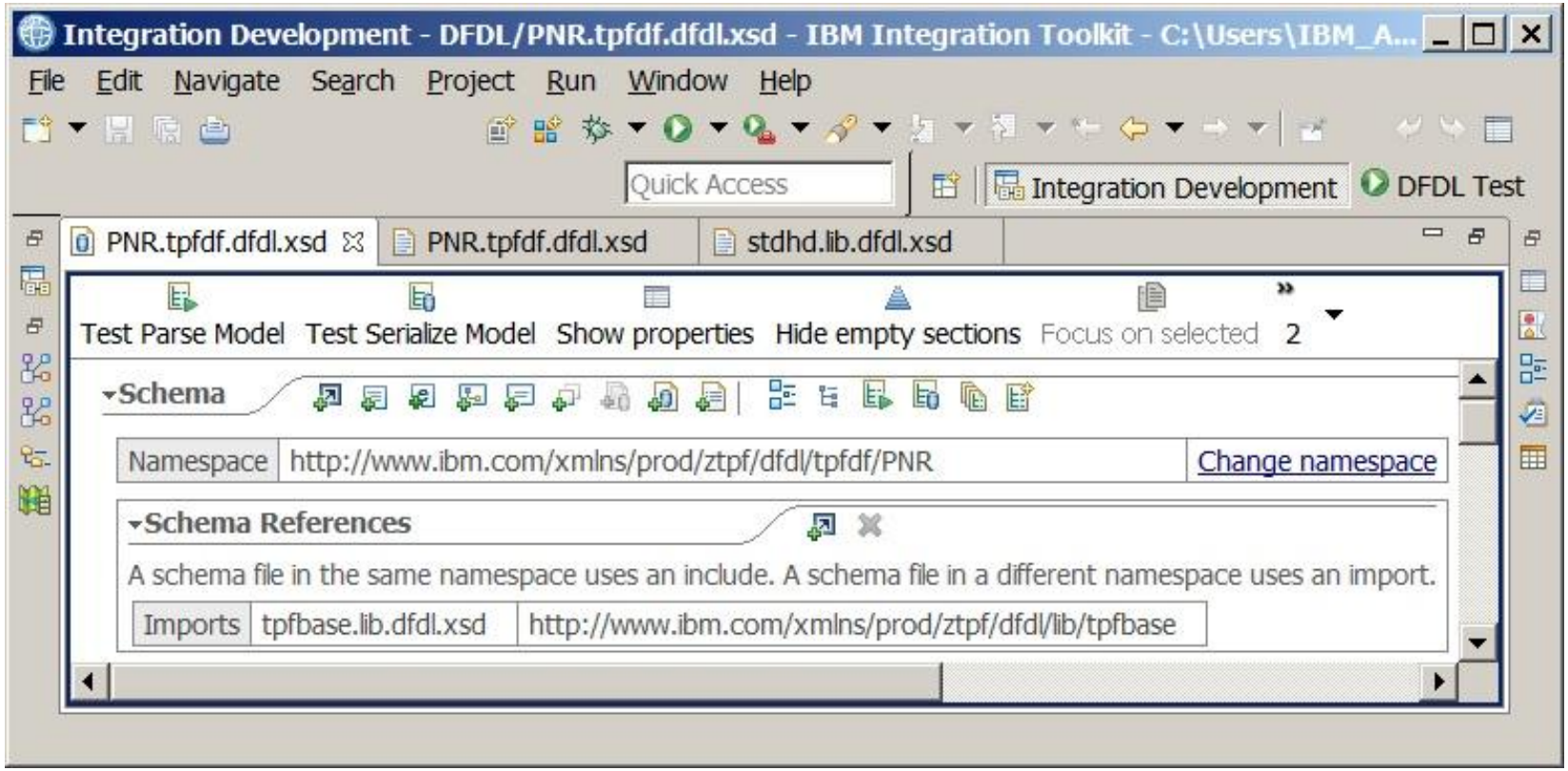
Import/Include Declarations

```
<xs:import namespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/lib/
tpfbase"
    schemaLocation="tpfbase.lib.dfdl.xsd" />
```

Locating Namespaces

- The import declaration associates an XML namespace with the file containing the definitions for that namespace.
- On z/TPF all DFDL files are loaded to a single directory so the location should not include any subdirectories.
- Imports are used for files that have different targetNamespaces while includes are used for files that share the same targetNamespace.

tpfbase import in the DFDL editor



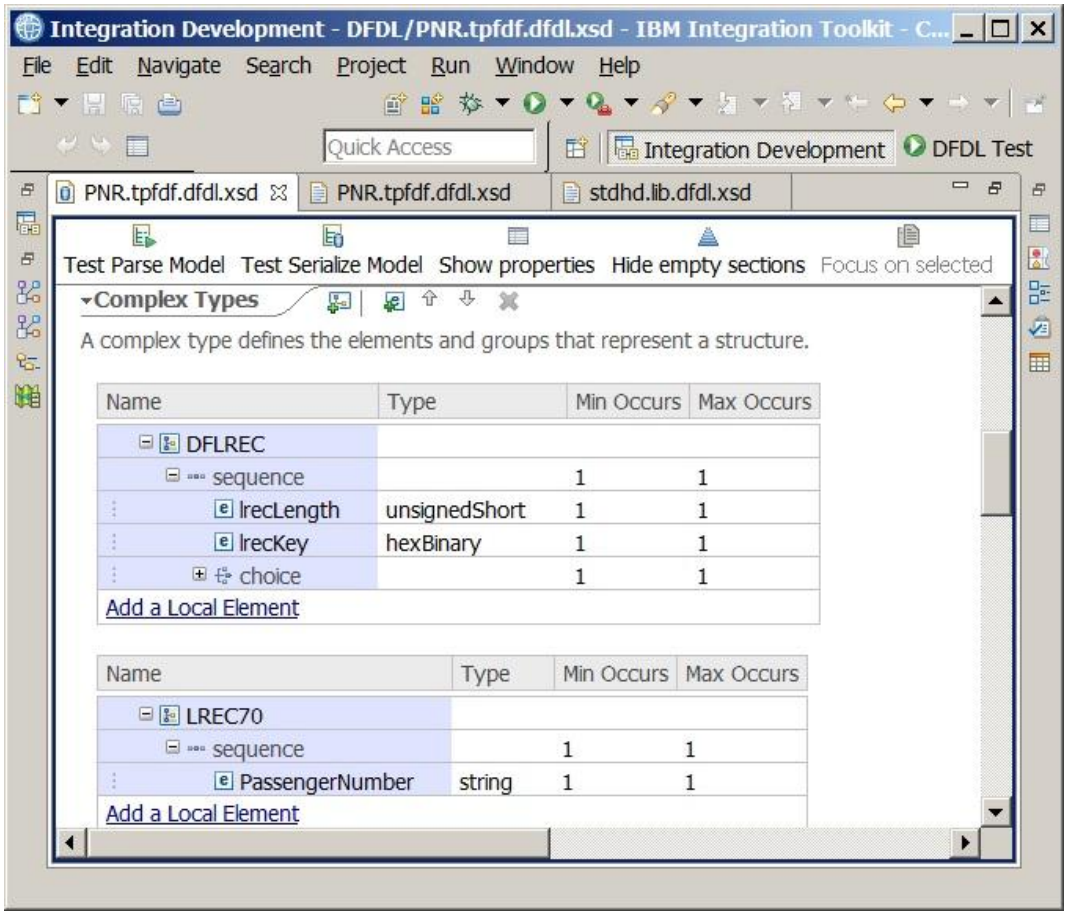
complexType Definitions

```
<xs:complexType name="LREC80" tddt:lrecId="80">
  <xs:sequence>
    <xs:element name="PassengerName" type="xsd:string"
        dfdl:length="25" tddt:offset="3"
        tddt:origName="DR26PNA" tddt:tpfType="C" />
  </xs:sequence>
</xs:complexType>
```

Defining Complex Data Types

- A complexType is similar to a C struct. Provides a way for referencing reusable data layouts.
- complexType names do not appear anywhere in an XML or JSON document and cannot be used as root elements. They are in essence names for internal data definition use only.
- Named complexType definitions can only appear at the top level of a schema definition.
- Element names and complexType names can be the same.

complexTypes in the DFDL editor



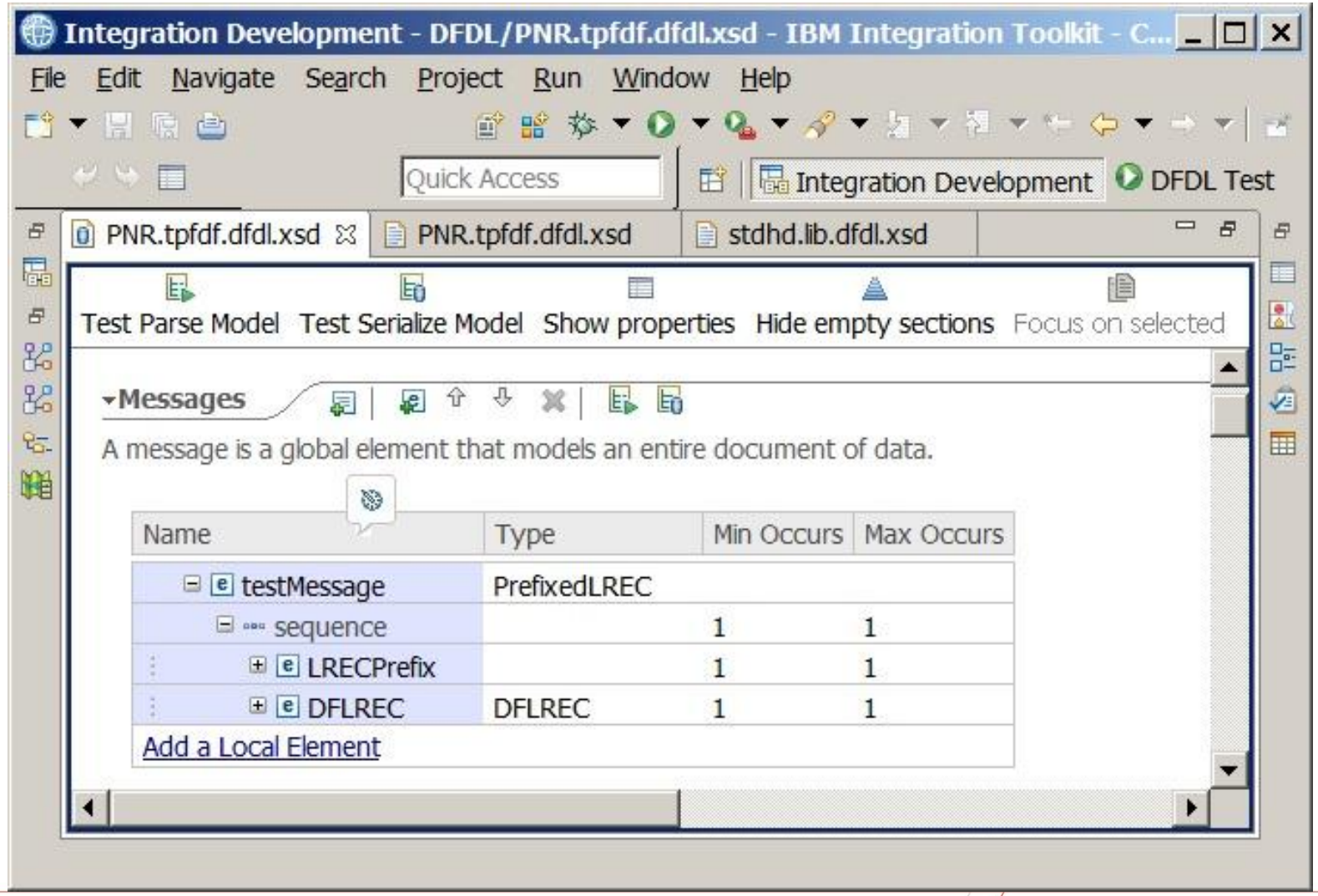
Root Element Definitions

```
<xs:element dfdl:lengthKind="implicit“ ibmSchExtn:docRoot="true“
    name="testMessage“ type="ns0:PrefixedLREC"/>
```

Root Element FAQ

- What is a root element?
  - A root element encompasses the body of the data description (or in XML terms, all other elements).
  - XML requires a root element.
- Do I need a root element if I'm not going to use an XML format of the data or if this file is just for defining complexTypes?
  - Yes, the DFDL editor uses this to test parsing and serialization.
  - Since DFDL is based on XML schema, a root element is assumed.
- Elements are either complex or simple, just like a field in C is either a struct or a primitive data type. Root elements are complex. Complex elements are composed of a “sequence” of elements and/or “choice” of elements.

root element of testMessage in the DFDL editor



DFDL Annotations

z/TPF supported DFDL annotations

- “long form” v “short form” annotations
- Element length
- Element occurrence
- Choice lengths
- Discriminators
- DFDL default values

“long form” v “short form”

- DFDL in “long form” (DFDL annotation):

```
<xs:element name="PassengerName" type="xs:string">
  <xs:annotation>
    <xs:appinfo source=="http://www.ogf.org/dfdl/">
      <dfdl:element dfdl:length="25"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

- DFDL in “short form” (DFDL attribute):

```
<xs:element name="PassengerName" type="xs:string“
 dfdl:length="25"/>
```

- Both have the exact same meaning.

Element Length Specification

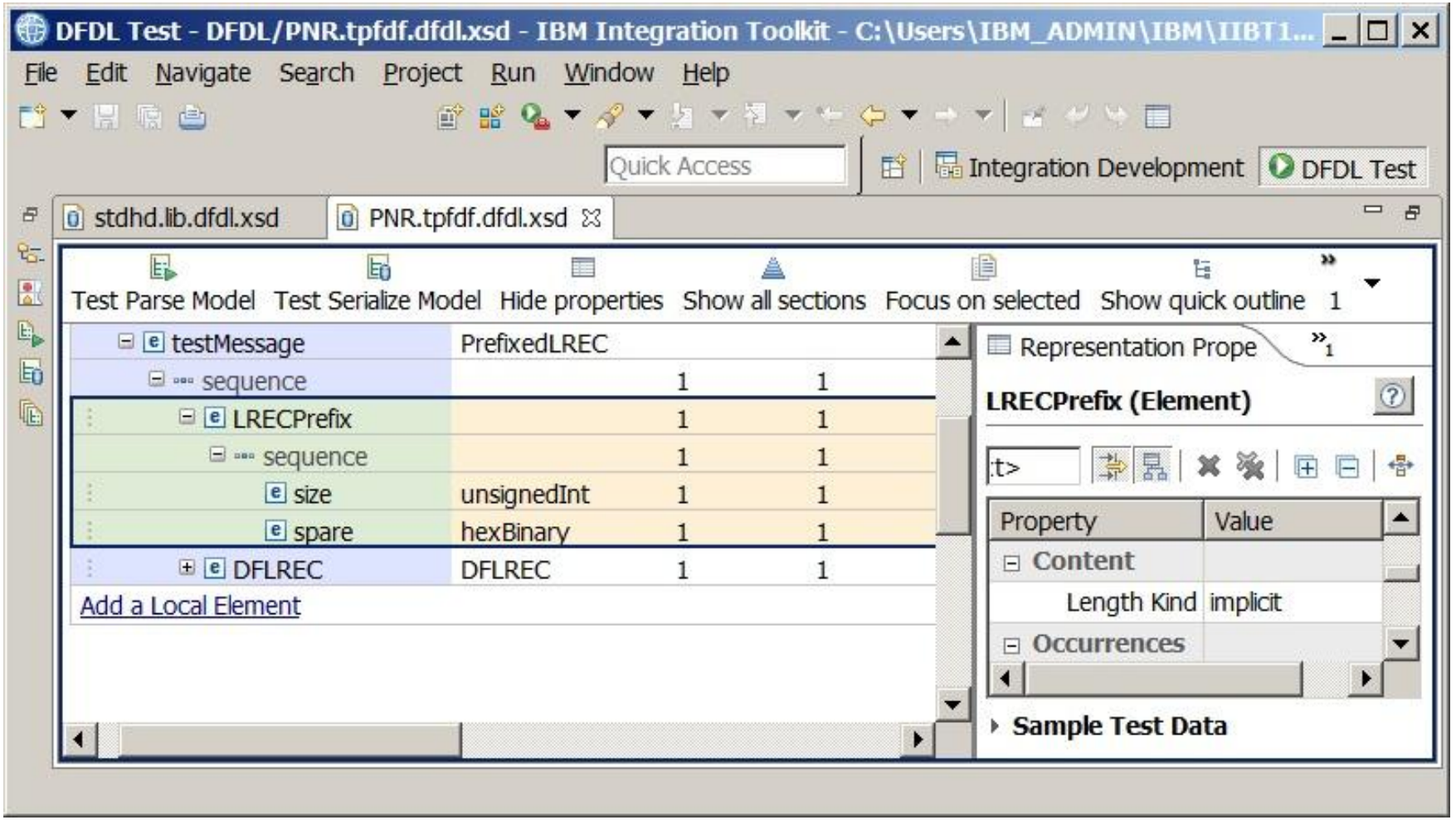
- z/TPF supports 2 ways of specifying an element length through the DFDL “lengthKind” attribute.
  - lengthKind=”explicit” means the DFDL attribute of “length” contains the length information. This is the default as found in tpfbase.lib.dfdl.xsd.
  - lengthKind=”implicit” means the length is inferred.

XSD data type	Inferred length (in bytes)
byte,unsignedByte	1
short,unsignedShort	2
int,unsignedInt,float,boolean	4
long,unsignedLong,double,date,time,dateTime	8
string,hexBinary	maxLength facet
decimal,integer, nonNegativeInteger	Not allowed

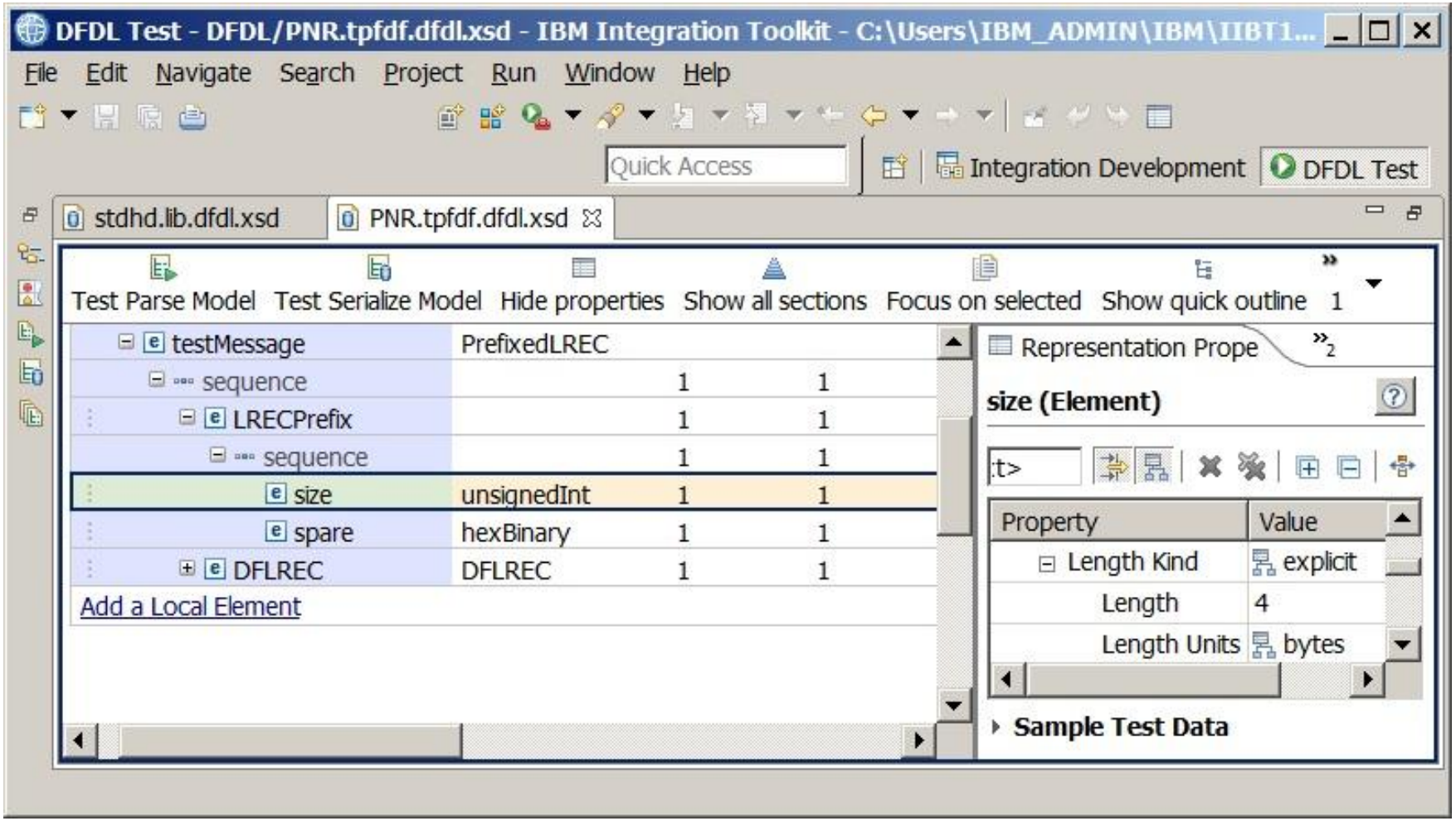
Element Length Example

```
<xs:element dfdl:lengthKind="implicit" name="LRECPrefix">
  <xs:complexType>
    <xs:sequence>
      <xs:element dfdl:length="4" name="size"
type="xs:unsignedInt"/>
      <xs:element dfdl:length="2" name="spare"
type="xs:hexBinary"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Implicit Length in DFDL editor



Explicit Length in DFDL editor



Element Length Units Specification

- z/TPF supports 2 units of length through the DFDL “lengthUnits” attribute.
  - lengthUnits=”bytes” means the length is specified in terms of number of bytes. This is the default as found in tpfbase.lib.dfdl.xsd.
  - lengthUnits=”bits” means the length is specified in terms of number of bits. Bit lengths can only be used for numeric types and boolean. Be careful using signed numeric types with a bit length field.

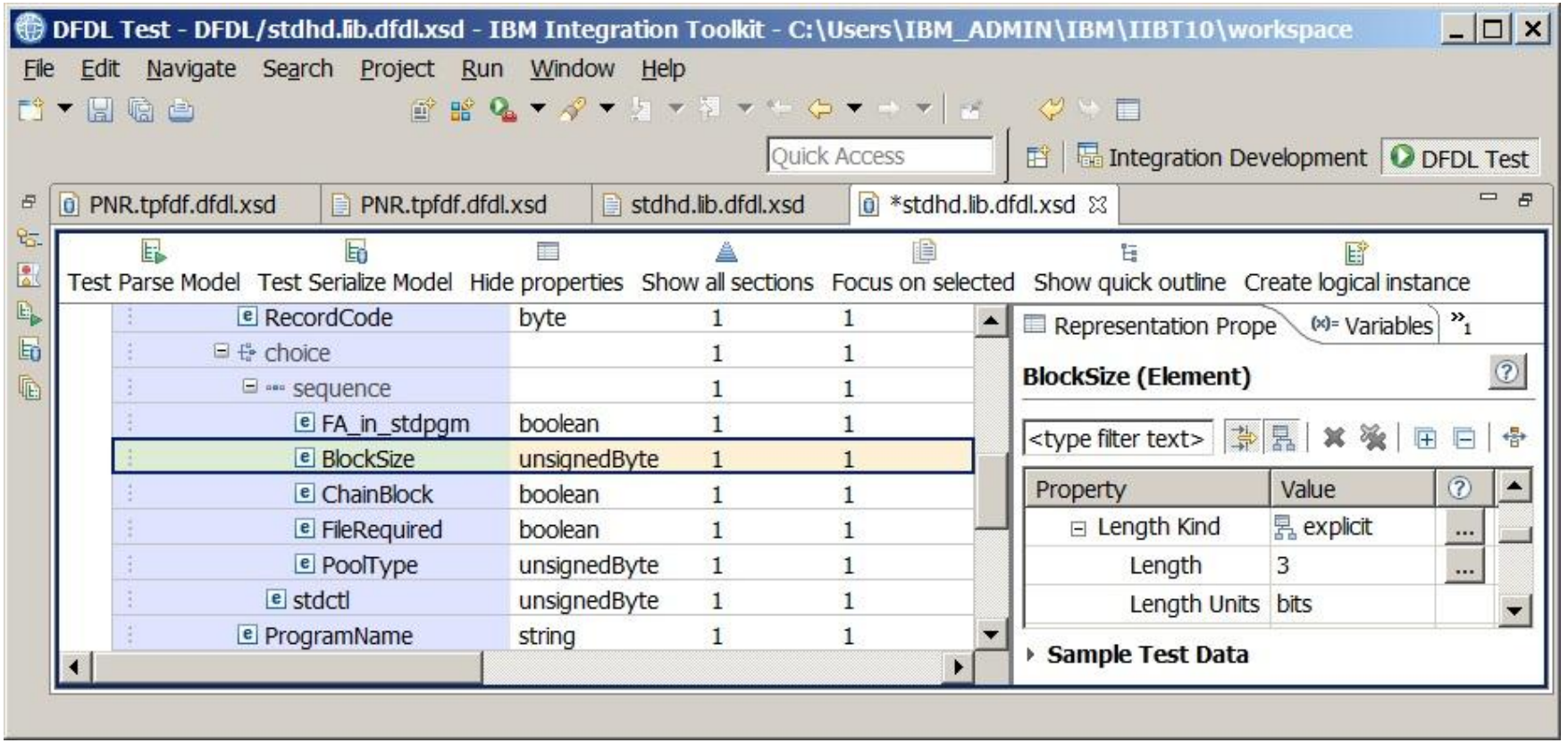
Element Length Units Example - DSECT

```
STDCTL&CG1 DS  B     CONTROL BYTE
*                    TPFDF BIT SETTINGS:
*                    BIT 0    0=ORDINAL NUMBER IN STDPGM
*                             1=FILE ADDRESS IN STDPGM OR
*                               HEX ZERO IF OFFL. CREATED
*                    BIT 1-3  VALUE BLOCKSIZE
*                             0=BLOCKSIZE  381
*                             1=BLOCKSIZE 1055
*                             2=BLOCKSIZE 4095
*                             3-7=IBM RESERVED
*                    BIT 4    0=PRIME BLOCK
*                             1=CHAIN BLOCK
*                    BIT 5    1=FILE REQUIRED
*                    BIT 6-7  VALUE POOLTYPE
*                             0=POOLTYPE 0
*                             1=POOLTYPE 1
*                             2=POOLTYPE 2
*                             3=IBM RESERVED
```

Element Length Units Example - DFDL

```
<xs:choice>
  <xs:sequence>
    <xs:element name="FA_in_stdpgm" type="xs:boolean"
        dfdl:length="1" dfdl:lengthUnits="bits"/>
    <xs:element name="BlockSize" type="xs:unsignedByte"
        dfdl:length="3" dfdl:lengthUnits="bits"/>
    <xs:element name="ChainBlock" type="xs:boolean"
        dfdl:length="1" dfdl:lengthUnits="bits"/>
    <xs:element name="FileRequired" type="xs:boolean"
        dfdl:length="1" dfdl:lengthUnits="bits"/>
    <xs:element name="PoolType" type="xs:unsignedByte"
        dfdl:length="2" dfdl:lengthUnits="bits"/>
  </xs:sequence>
  <xs:element name="stdctl" type="xs:unsignedByte"
      dfdl:length="1"/>
</xs:choice>
```

Length Units in the DFDL editor



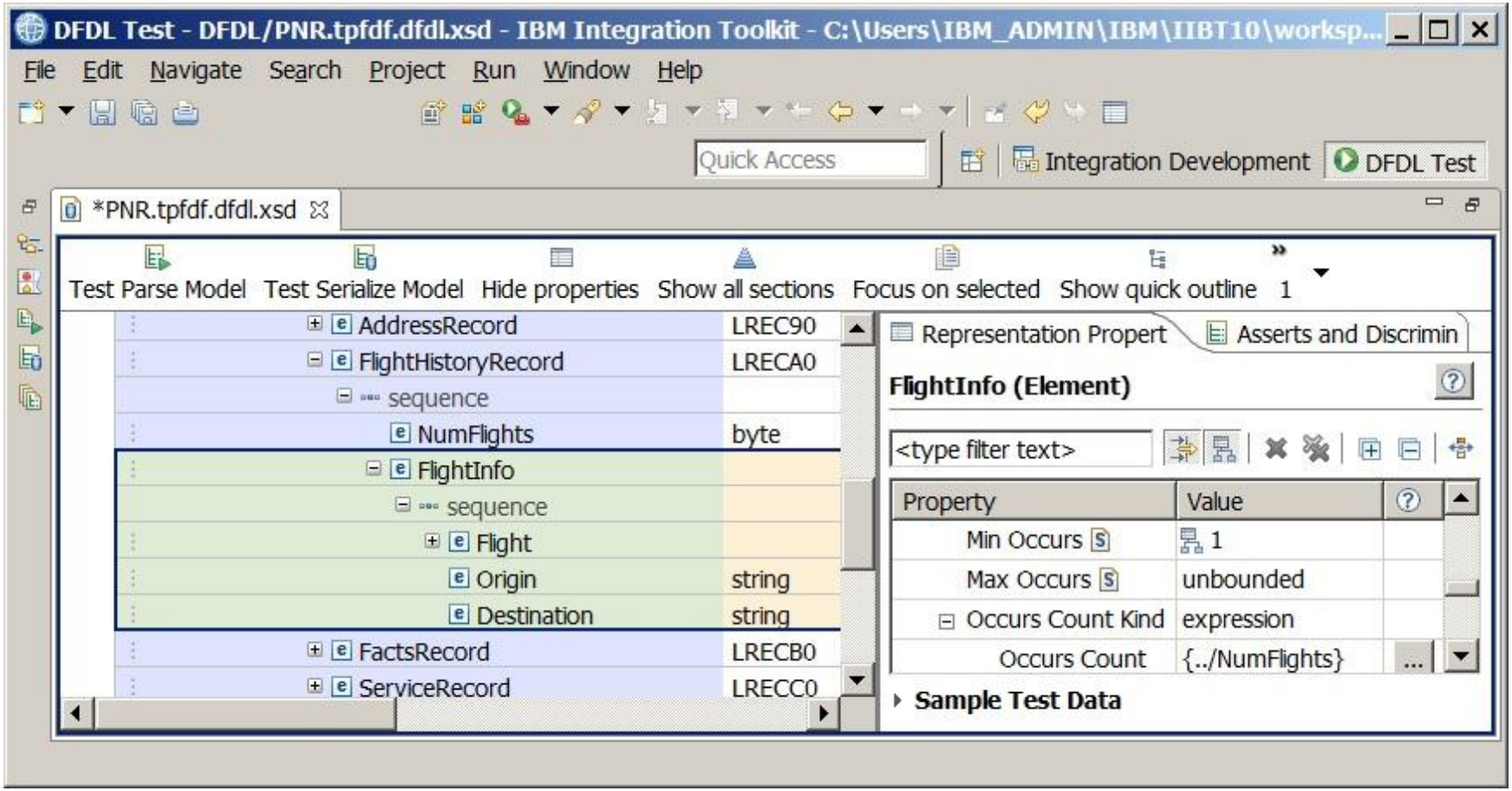
Element Occurrence Specification

- z/TPF supports 2 ways of specifying an element occurrence (array size) through the DFDL “occursCountKind” attribute.
  - occursCountKind=”fixed” means the minOccurs and maxOccurs attributes are equal and contain the number of occurrences. This is the default as found in tpfbase.lib.dfdl.xsd. minOccurs and maxOccurs both have default values of 1.
  - occursCountKind=”expression” means the DFDL attribute “occursCount” contains the occurrence information. minOccurs and maxOccurs specify the lower and upper boundaries respectively.

Element Occurrence Example

```
<xs:element dfdl:length="1" name="NumFlights"
    tddt:offset="3" tddt:origName="DR26SP1" tddt:tpfType="C"
    type="xs:byte"/>
<xs:element dfdl:length="11"
    dfdl:occursCount="{../NumFlights}"
    dfdl:occursCountKind="expression" maxOccurs="unbounded"
    name="FlightInfo" tddt:offset="4"
    tddt:origName="DR26FLI">
```

Element Occurrence in the DFDL editor



Choice Length Specification

- The DFDL “choiceLengthKind” attribute works similarly to the DFDL “lengthKind” attribute however it can only be used on a choice declaration.
  - Instead of using the DFDL “length” attribute, there's a DFDL “choiceLength” attribute
  - The units of length is always in bytes
  - choiceLengthKind of “implicit” is the default
- Specifying a length for a choice better resembles how a union would work in C or how an “ORG ,” would work in assembler.

Choice Length Example

The following C union

```
union {
    char small;       /* 1 byte field */
    int medium;       /* 4 byte field */
    long large;       /* 8 byte field */
}
```

could become

```
<xs:choice dfdl:choiceLengthKind="explicit" dfdl:choiceLength="8">
  <xs:element name="small" type="xs:byte" dfdl:length="1"/>
  <xs:element name="medium" type="xs:int" dfdl:length="4"/>
  <xs:element name="large" type="xs:long" dfdl:length="8"/>
</xs:choice>
```

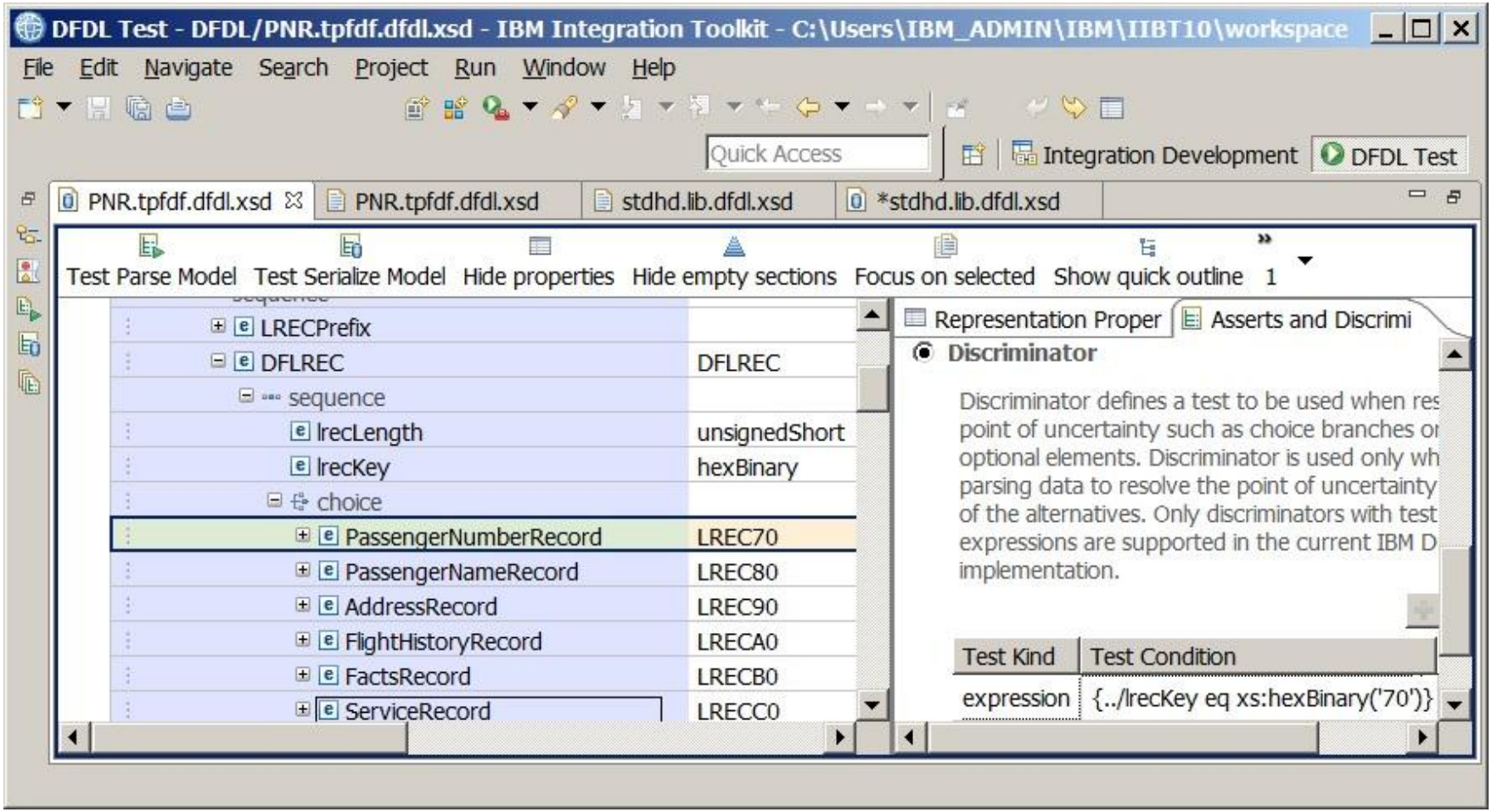
Discriminator Annotations

- Discriminators are a true/false test of existence for some component of the logical format of the data.
- Discriminators can be used on element, sequence, choice, and simpleType declarations.
- Discriminators are used only during parsing, not serialization. They are not really meant to validate data.

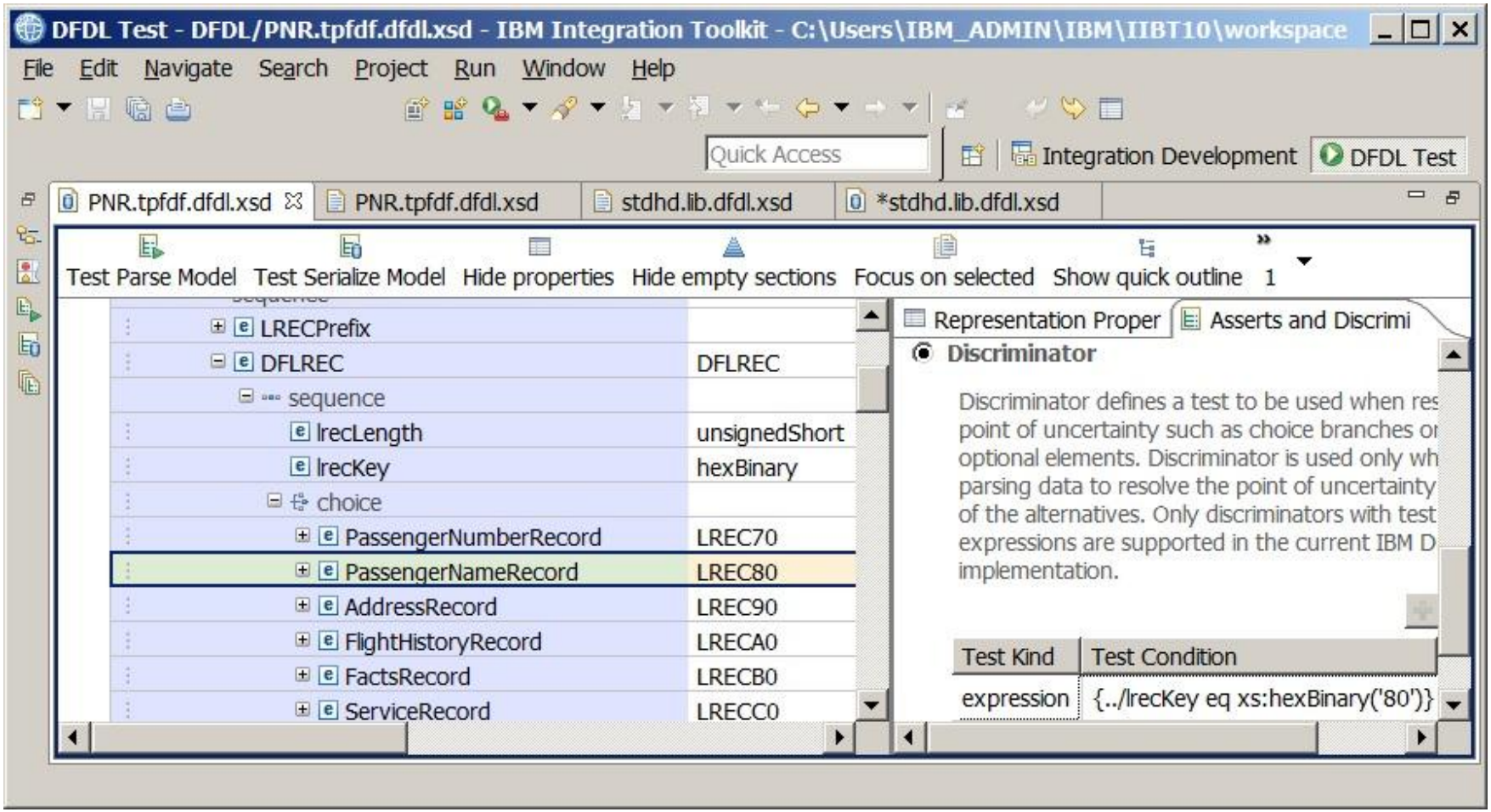
Discriminator Example

```
<xs:element dfdl:length="1" name="lrecKey" type="xs:hexBinary"/>
<xs:choice>
  <xs:element dfdl:lengthKind="implicit"
      name="PassengerNumberRecord" type="ns1:LREC70">
    <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
        <dfdl:discriminator>{../lrecKey eq xs:hexBinary('70')}
        </dfdl:discriminator>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
  <xs:element dfdl:lengthKind="implicit"
      name="PassengerNameRecord" type="ns1:LREC80">
    <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
        <dfdl:discriminator>{../lrecKey eq xs:hexBinary('80')}
        </dfdl:discriminator>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
```

Discriminator for PassengerNumberRecord



Discriminator for PassengerNameRecord



DFDL Defaults (tpfbase.lib.dfdl.xsd)

- DFDL attribute default values are currently hard coded on z/TPF.
- tpfbase.lib.dfdl.xsd contains the equivalent default value settings in a DFDL “format” annotation for portability to other platforms (as well as DFDL editor).
- Summary: Don't change tpfbase.lib.dfdl.xsd

DFDL Expressions

DFDL Expressions

- Expression syntax
- Expression types
- Literals
- Node (element) references
- Type casting

DFDL Expression Syntax

```
DFDL Expression ::= “{“ Expr “}”
Expr            ::= ExprSingle
ExprSingle      ::= IfExpr | OrExpr
IfExpr          ::= “if” “(“ Expr “)” “then” ExprSingle “else” ExprSingle
OrExpr          ::= AndExpr ( “or” AndExpr )*
AndExpr         ::= ComparisonExpr ( “and” ComparisonExpr )*
ComparisonExpr  ::= AdditiveExpr ( (ValueComp) AdditiveExpr )?
AdditiveExpr    ::= MultiplicativeExpr ( (“+” | “-”) MultiplicativeExpr )*
MultiplicativeExpr ::= ValueExpr ( (“*” | “idiv” | “mod”) ValueExpr )*
ValueExpr       ::= PathExpr
ValueComp       ::= “eq” | “ne” | “lt” | “le” | “gt” | “ge”
PathExpr        ::= (“/” RelativePathExpr?) | RelativePathExpr
RelativePathExpr ::= StepExpr ( “/” StepExpr )*
StepExpr        ::= FilterExpr | AxisStep
AxisStep        ::= ReverseStep | ForwardStep
```

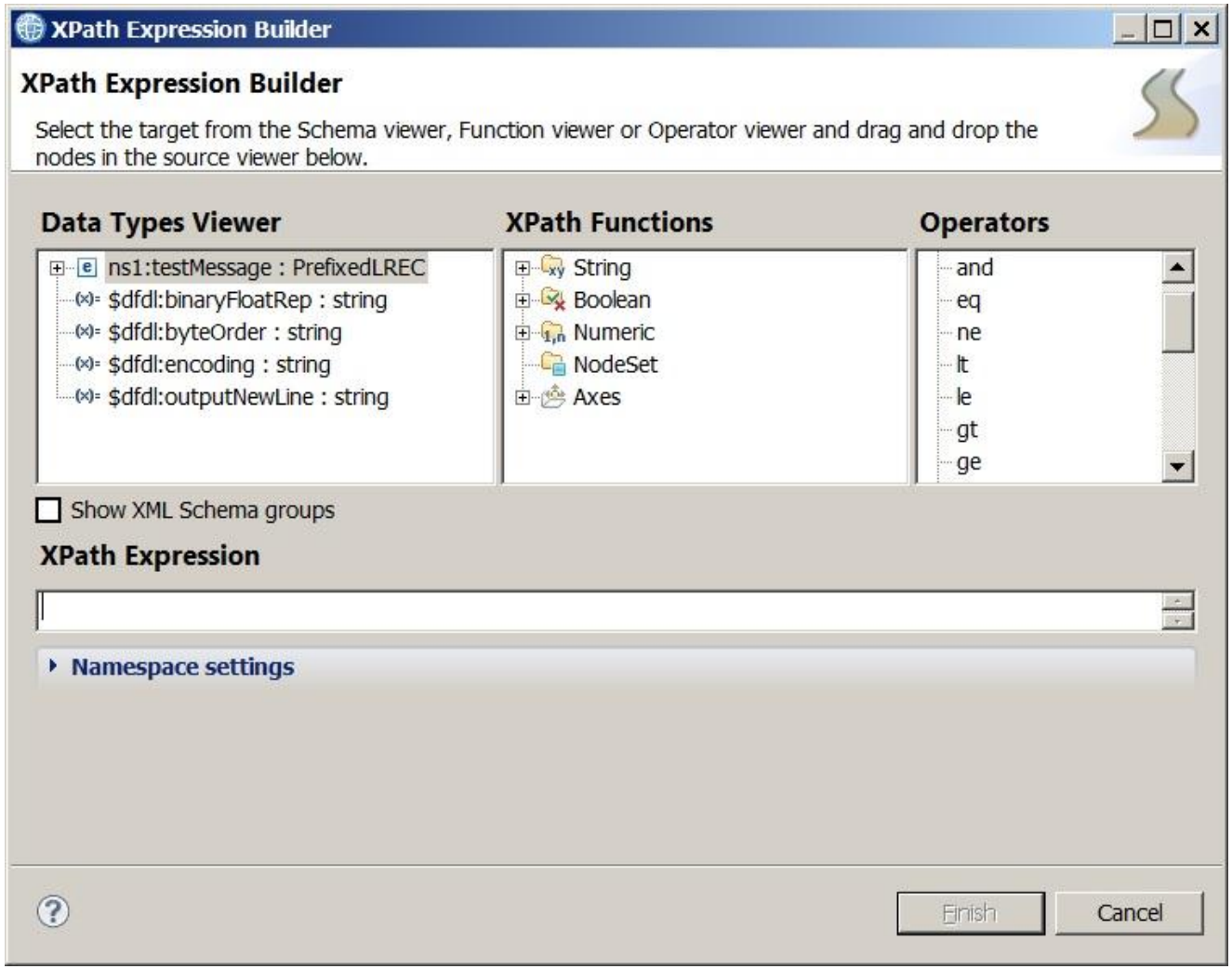
DFDL Expression Syntax (cont'd)

```
ForwardStep       ::= AbbrevForwardStep
AbbrevForwardStep ::= NodeTest
ReverseStep       ::= AbbrevReverseStep
AbbrevReverseStep ::= ".."
NodeTest          ::= NameTest
NameTest          ::= Qname
FilterExpr        ::= PrimaryExpr
PrimaryExpr       ::= Literal | ParenthesizedExpr | FunctionCall
Literal           ::= NumericLiteral | StringLiteral
NumericLiteral    ::= IntegerLiteral
ParenthesizedExpr ::= "(" Expr? ")"
FunctionCall      ::= Qname "(" ExprSingle ")"
Qname             ::= ([A-Za-z_][A-Za-z0-9_-]*[:])? [A-Za-z][0-9A-Za-z_-]*
IntegerLiteral    ::= ("-" | "+")? [0-9]+
StringLiteral     ::= ('"' [^"]* '"') | ("'" [^']* "'")
```

DFDL Expressions

- DFDL uses a subset of XPath expressions.
- DFDL expressions are enclosed within curly braces: “{“ “}”
- Only DFDL annotations/attributes can contain DFDL expressions, not normal XML schema components.
  - dfdl:length
  - dfdl:occursCount
  - dfdl:choiceLength
  - dfdl:discriminator
- A DFDL Expression Builder is found in the DFDL editor using CTRL+SPACE.

Expression Builder in DFDL editor



DFDL Expression Types

- conditional expressions (if,then,else)
  - equivalent to C except there is always an else
- logical expressions (and,or)
  - equivalent to &&,|| in C
  - these are not bitwise operations
- relational expressions (eq,ne,lt,le,gt,ge)
  - equivalent to ==,!=,<,<=,>,>= in C
- arithmetic expressions (+,-,*,idiv,mod)
  - equivalent to integer arithmetic in C of +,-,*,/,%

DFDL Literals

- z/TPF supports 2 types of literals:
  - Integer literal
    - Example:

```
<xs:element name="Version" type="xs:byte"
dfdl:length="1"/>
<xs:element name="Name" type="xs:string" dfdl:length="{if
(../Version gt 2) then 16 else 8}"
```

  - String literal
    - These are enclosed in either single (‘) or double (“) quotes.
    - Example:

```
<xs:element name="Code" type="xs:string" dfdl:length="1"/>
<xs:element name="Name" type="xs:string" dfdl:length="{if
(../Code eq "Q") then 16 else 8}
```

Node References (XPath location paths)

- Node references work very similar to how hfs directories are navigated. Elements can be thought to be the equivalent of a directory names.
- Two methods of locating an element:
  - Absolute path: /<root element>/<path>
  - Relative path: ../<path> or ./<path> preferred method
    - ".." references the parent element in the tree
      - This is used in DFDL expressions for discriminators and DFDL attributes on element definitions.
    - "." references the currently processed element in the tree
      - This is used in DFDL expressions for discriminators on sequence and choice declarations.

Parent Item (“..”) example

```
<xs:element dfdl:length="1" name="lrecKey" type="xs:hexBinary"/>
  <xs:choice>
    <xs:element dfdl:lengthKind="implicit"
        name="PassengerNumberRecord" type="ns1:LREC70">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:discriminator>{../lrecKey eq xs:hexBinary('70')}
          </dfdl:discriminator>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
```

Context Item (“.”) example

```
<xs:complexType name="LRECC0" tddt:lrecId="C0">
  <xs:sequence>
    <xs:element dfdl:length="1" name="ServiceCode"
        tddt:offset="3" tddt:origName="DR26SVC" tddt:tpfType="X"
        type="xs:byte"/>
      <xs:choice>
        <xs:sequence>
          <xs:annotation>
            <xs:appinfo source="http://www.ogf.org/dfdl/">
              <dfdl:discriminator message="">{./ServiceCode eq 1}
              </dfdl:discriminator>
            </xs:appinfo>
          </xs:annotation>
```

Type casting

- Comparison expressions require the same data type for the compare.
- DFDL expressions create casts through constructor functions.
- z/TPF supports the following type casts in DFDL:
  - numeric types
  - boolean
  - hexBinary
- Example:

```
{../lrecKey eq xs:hexBinary('80')}
```

casts the string literal ‘80’ to a hexBinary value of 0x80 to compare to the data in element lrecKey.

References

- DFDL tutorials created by the DFDL Working Group at the Open Grid Forum
  - http://redmine.ogf.org/dmsf/dfdl-wg?folder_id=5485
- DFDL developerWorks tutorials
  - http://ibm.biz/startdfdl
- DFDL specification reference
  - http://www.ogf.org/dfdl

Trademarks

- IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.
- *(Include any special attribution statements as required – see Trademark guidelines on https://w3-03.ibm.com/chq/legal/lis.nsf/lawdoc/5A84050DEC58FE31852576850074BB32?OpenDocument#Developing%20the%20Special%20Non-IBM%20Tr)*

Notes

- Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
- All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.
- This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
- All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
- Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
- Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.
- This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.