XML-INTO options for namespaces and other non-RPG names

- There are two new XML-INTO options to handle XML names containing a namespace.
    - The namespace option, ns, controls how XML-INTO handles XML names with a namespace when it is matching XML names to the names in the path option or the subfield names of a data structure.
    - The namespace prefix option, nsprefix, allows your RPG program to find out the values of the namespaces that were removed from the XML names when the *ns=remove* option was used to remove the namespace from the names.
- Also, there is a new value for the case option, case=convert, to handle non-RPG characters in XML names

ns (default *keep*)

The *ns* option controls how XML-INTO handles XML names with a namespace when XML-INTO is matching XML names to the names in the *path* option or the subfield names of a data structure. For example, the XML name "cust:name" has the namespace "cust".

- indicates that the namespace and colon are retained in the XML name. An XML name with a namespace will not match any RPG name.

- *remove* indicates that the namespace and colon are removed from the XML name when matching an RPG name. For example, if the XML name is *ABC:DEF*, the name *DEF* is used when comparing to an RPG name.

- *merge* indicates that the colon is replaced with underscore in the XML name when matching an RPG name. For example, if the XML name is *ABC:DEF*, the name *ABC_DEF* is used when comparing to an RPG name.

Notes:

1. The *ns* option is in effect when handling the *path* option. The names in the path must be specified so that they will match the XML names after the processing for the ns option. For example, if an XML path is *abc:info/abc:cust* and option *'ns=remove'* is specified, then the *path* option must be specified as 'path=info/cust'. If option *ns=merge* is specified, then the *path* option must be specified as 'path=abc_info/abc_cust'.

2. If option *ns=remove* is specified, the *nsprefix* option can be used to get the value of the namespace for any subfield.

Examples of the *ns* option

1. Option *ns=remove* is used.

   The following definition is used in the example

```
D info            DS                  QUALIFIED
D   type                         25A   VARYING
D   qty                          10I 0
D   price                         7P 3
```

   Assume that file **info1.xml** contains the following

```
<abc:info xmlns:abc="http://www.abc.xyz">
 <abc:type>Chair</abc:type>
 <abc:qty>3</abc:qty>
 <abc:price>79.99</abc:price>
</abc:info>
```

   The names in the XML document, such as *abc:type*, cannot be used for RPG subfield names.

   Option *ns=remove* specifies that the namespace (*abc*) and colon should be removed from the XML name before the XML-INTO operation searches for a matching subfield or for a name specified in the *path* option.

   The XML name *abc:type* matches the RPG subfield *TYPE* after the namespace *abc:* is removed.

```
xml-into info %xml('info1.xml'
                   : 'doc=file ns=remove');
// info.type = 'Chair'
// info.qty = 3
// info.price = 79.99
```

2. Option *ns=merge* is used.

   The following definition is used in the example

```
D info            DS                  QUALIFIED
D   abc_type                     25A   VARYING
```

```
D   def_type                      25A   VARYING
D   abc_qty                       10I 0
D   abc_price                      7P 3
```

Assume that file **info2.xml** contains the following

```
<abc:info xmlns:abc="http://www.abc.xyz"
          xmlns:def="http://www.def.xyz">
 <abc:type>Chair</abc:type>
 <abc:qty>3</abc:qty>
 <def:type>Modern</def:type>
 <abc:price>79.99</abc:price>
</abc:info>
```

The XML document contains name *abc:type*, with namespace *abc*.

The XML document also contains name *def:type*, with namespace *def*. Option *namespace=remove* cannot be used in this case, because there would be two different XML elements whose names would match an RPG subfield *TYPE*.

Option *ns=merge* is specified, indicating that the namespace (*abc*) should be merged with the remainder of the XML name, with an underscore separating the two parts of the name, before the XML-INTO operation searches for a matching subfield.

The name *abc_type* matches the RPG subfield *ABC_TYPE* after the namespace *abc* is merged with *type*. The name *def_type* matches the RPG subfield *DEF_TYPE* after the two parts of the XML name are merged.

The data structure name *info* does not match the merged XML name *abc_info*, so the path option must be specified. The merged name *abc_info* is used in the path option.

See namespace prefix option for another way to handle this type of XML document.

```
xml-into info %xml('info2.xml'
                   : 'doc=file ns=merge path=abc_info');
// info.abc_type = 'Chair'
// info.def_type = 'Modern'
// info.abc_qty = 3
// info.abc_price = 79.99
```

nsprefix

The *nsprefix* option allows your RPG program to determine the values of the namespaces that were removed from the XML names when option *ns=remove* was specified.

The *nsprefix* option specifies the prefix for the names of the subfields that are to receive the value of the namespace. The *nsprefix* option is ignored unless option *ns=remove* is specified.

For example, if the XML element <abc:def>hello</abc:def>, and options *ns=remove* and *nsprefix=PFX_* are specified, then RPG subfield *DEF* will receive the value "hello" and RPG subfield *PFX_DEF* will receive the value "abc".

Rules for the *nsprefix* option:

1. The *nsprefix* subfield must have alphanumeric or UCS-2 type.

2. If a subfield matched by XML data is an array, the *nsprefix* subfield must also be an array, with the same number of elements. If a subfield matched by XML data is not an array, the *nsprefix* subfield must not be an array.

3. If an XML element does not have a namespace, the empty string '' will be placed in the *nsprefix* subfield.

4. It is not considered an error if a subfield has the correct name for an *nsprefix* subfield but it does not meet the criteria for being an *nsprefix* subfield. For example, if *nsprefix=ns* is specified, and the data structure has array subfield *NAME* with two elements, and it has alphanumeric array subfield *NSNAME* with three elements, the subfield *NSNAME* is not considered to be an *nsprefix* subfield, so XML-INTO will expect to find XML data to set its value.

5. The *case* option does not affect the namespace value that is placed in the *nsprefix* subfield. For example, if the *case=convert* option is specified, and the XML name is *a--b:name*, the value "a--b" will be placed in the *nsprefix* subfield.

6. The *nsprefix* option is not considered for the *datasubf* subfield.

Example of the *nsprefix* option

1. The following definition is used in the example

```
D info            DS                  QUALIFIED
D   type                         25A   VARYING DIM(2)
D   ns_type                      10A   VARYING DIM(2)
D   qty                          10I 0
D   price                         7P 3
D   ns_price                     10A   VARYING
```

   Assume that file info3.xml contains the following

```
<abc:info xmlns:abc="http://www.abc.xyz"
          xmlns:def="http://www.def.xyz">
 <abc:type>Chair</abc:type>
 <abc:qty>3</abc:qty>
 <def:type>Modern</def:type>
 <abc:price>79.99</abc:price>
</abc:info>
```

   XML-INTO options *ns=remove nsprefix=ns_* are specified, so that the RPG programmer can obtain the namespace used for the XML name matching some of the RPG subfields. Option *nsprefix=ns_* indicates that subfields beginning with *NS_* are candidates for holding the namespace values.

   The XML document has two elements that map to the RPG subfield *TYPE*, *abc:type* and *def:type*.

   The *TYPE* subfield is defined with **DIM(2)** because there are two XML elements with the name *type*, after the namespace is removed. The *NS_TYPE* subfield is also defined with **DIM(2)** so that XML-INTO can place the namespace value for each occurrence of an XML name matching the *TYPE* subfield.

   When XML-INTO handles the XML name *abc:type*, it will set the *TYPE(1)* subfield to the value 'Chair' and it will set the *NS_TYPE(1)* subfield to the value 'abc'.

   When XML-INTO handles the XML name *def:type*, it will set the *TYPE(2)* subfield to the value 'Modern' and it will set the *NS_TYPE(2)* subfield to the value 'def'.

When XML-INTO handles the XML name *abc:qty*, it sets the *QTY* subfield to the value 3. There is no subfield with the name *NS_QTY*, so the namespace value is not saved in a subfield.

When XML-INTO handles the XML name *abc:price*, it will set the *PRICE* subfield to the value 79.99 and it sets the *NS_PRICE* subfield to the value 'abc'.

```
xml-into info %xml('info3.xml'
                   : 'doc=file ns=remove nsprefix=ns_');
// info.type(1) = 'Chair'
// info.ns_type(1) = 'abc'
// info.type(2) = 'Modern'
// info.ns_type(2) = 'def'
// info.qty = 3
// info.price = 79.99
// info.ns_price = 'abc'
```

case (default *lower*)

The *case* option specifies the way that XML-INTO should interpret the element and attribute names in the XML document when searching for XML names that match the the RPG field names and the names in the *path* option. If the XML elements are not interpreted correctly, they will not be successfully matched to the subfield names and the names in the path, and the operation will fail with status 00353.

- indicates that the XML element and attribute names matching the RPG variable names are in lower case.

- *upper* indicates that the XML element and attribute names matching the RPG variable names are in upper case.

- *any* indicates that the element and attribute names matching the RPG variable names are in unknown or mixed case. The XML element and attribute names will be converted to upper case before comparison to the upper-case RPG variable names.

- *convert* indicates that the names in the XML document are converted to valid RPG names before matching to RPG names. The name is converted by the following steps:

    1. The alphabetic characters in the name are converted to the uppercase A-Z characters using the ***LANGIDSHR** conversion table for the job. For example, an XML name "èñ-Àúb" would be converted to "EN-AUB" in this step.

    2. Any characters in the name that are not A-Z and 0-9 after this step, including any double-byte character sequences in the name, are converted to the underscore character. For example, the name "EN-AUB#" will be converted to "EN_AUB_" during this step.

    3. Any remaining underscores are merged into a single underscore. This includes both underscores that appear in the original name, and

underscores that have been added in the previous steps. For example, the name "EN-$_AUB" would have been converted to "EN___AUB" in the previous step, and it would be converted to "EN_AUB" in this step.

4. If the first character in the resulting name is the underscore character, it is removed from the name. For example, the name "_EN_AUB" will be converted to "EN_AUB" during this step.

5. Warning: Some alphabetic characters may not be converted to A-Z characters. For example the character 'Ä' is a separate character from A in the Swedish character set, so it does not map to character 'A' using the ***LANGIDSHR** conversion table. In a Swedish job, the XML name 'ABÄC' would not be changed during the first step of the conversion, so the 'Ä' character would still remain in the name after the first step. The 'Ä' character would be changed to _ during the second step, so the resulting name would be 'AB_C' rather than the 'ABAC' name which might be expected.

Examples of the *case=convert* option

1. The XML document contains names with alphabetic characters that are not valid characters for RPG subfields.

   The following data structures are used in the example

```
D etudiant        ds                  qualified
D  age                                3p 0
D  nom                               25a   varying
D  ecole                             50a   varying

D student         ds                  likeds(etudiant)
```

   Assume that file **info.xml** contains the following lines:

```
<Étudiant Nom="Élise" Âge="12">
   <École>Collège Saint-Merri</École>
</Étudiant>
```

   a. Options *case=convert ccsid=ucs2* are specified. Option *case=convert* specifies that the names in the XML document will be converted using the ***LANGIDSHR** translation table for the job before matching to the RPG names in the path and in the list of subfields. The names *Étudiant*, *Âge*, and *École* will be converted to *ETUDIANT*, *AGE*, and *ECOLE*. The XML data itself is not converted, so the subfield *ecole* will receive the value "Collège Saint-Merri" as it appears in the XML document.

      The path option is not necessary, because the default path is the name of the RPG variable *ETUDIANT*, which matches the converted form of the actual XML name, *Étudiant*.

```
xml-into etudiant %xml('info.xml'
                       : 'doc=file case=convert '
                       + 'ccsid=ucs2');
// etudiant.nom = 'Élise'
// etudiant.age = 12
// etudiant.ecole = 'Collège Saint-Merri'
```

   b. The RPG data structure is called *student*. The path option must be specified to indicate that the *Étudiant* XML element matches the *student* data structure. The path option is specified as *path=etudiant*, to match the XML name after conversion.

```
xml-into student %xml('info.xml'
                      : 'doc=file case=convert '
                      + 'ccsid=ucs2 path=etudiant');
// student.nom = 'Élise'
// student.age = 12
   // student.ecole = 'Collège Saint-Merri'
```

2. The XML document contains names with non-alphanumeric characters that XML supports but that cannot be used in RPG names.

   The following data structures are used in the examples

```
D employee_info   ds                  qualified
D  last_name                   25a    varying
D  first_name                  25a    varying
D  is_manager                   1a

D emp             ds                  likeds(employee_info)
```

   Assume that file **data.xml** contains the following lines:

```
<employee-info is-manager="y">
   <last-name>Smith</last-name>
   <first-name>John</first-name>
</employee-info>
```

   a. Option *case=convert* is specified. After any conversion of the alphabetic characters using the ***LANGIDSHR** table for the job, the next step converts any remaining characters that are not A-Z or 0-9 to the underscore character. XML names *employee-info*, *is-manager*, *last-name*, and *first-name* are converted to *EMPLOYEE_INFO*, *IS_MANAGER*, *LAST_NAME*, and *FIRST_NAME*.

      The RPG data structure name *employee_info* matches the converted form, *EMPLOYEE_INFO*, of the XML name *employee-info*, so the path option is not required.

```
xml-into employee_info %xml('data.xml'
                     : 'doc=file case=convert ');
// employee_info.last_name = 'Smith'
// employee_info.first_name = 'John'
// employee_info.is_manager = 'y'
```

b. The RPG data structure is called *emp*. The path option must be specified to indicate that the *employee-info* XML element matches the *emp* data structure. The path option is specified as *path=employee_info*, to match the XML name after conversion.

```
xml-into emp %xml('data.xml'
                  : 'doc=file case=convert '
                  + 'ccsid=ucs2 path=employee_info' );
// emp.last_name = 'Smith'
// emp.first_name = 'John'
// emp.is_manager = 'y'
```

3. The XML document contains names with double-byte data.

The following definitions are used in the examples

```
D employee_info_  ds                    qualified
D  last_name_                    25a    varying
D  first_name_                   25a    varying
D  is_manager_                    1a
```

Assume that file **data.xml** contains the following lines, where "DBCS" represents double-byte data:

```
<employee_info_DBCS is_manager_DBCS="y">
   <last_name_DBCS>Smith</last_name_DBCS>
   <first_name_DBCS>John</first_name_DBCS>
</employee_info_DBCS>
```

Option *case=convert* is specified. After any conversion of the alphabetic characters using the ***LANGIDSHR** table for the job, the next step converts any remaining characters that are not A-Z or 0-9 to the underscore character, including **DBCS** data and the associated shift-out and shift-in characters. After this step, the XML name *last_name_DBCS* would be converted to *LAST_NAME_____*. The next step merges any remaining underscores, including any underscores that appeared in the original name, to a single underscore. The resulting name is *LAST_NAME_*.

```
xml-into employee_info_ %xml('data.xml'
                     : 'doc=file case=convert '
                     + 'ccsid=ucs2');
// employee_info_.last_name_ = 'Smith'
// employee_info_.first_name_ = 'John'
// employee_info_.is_manager_ = 'y'
```

4. The XML document contains names with double-byte data at the beginning of the name.

   The following definitions are used in the examples

```
D employee_info   ds                   qualified
D  last_name                     25a   varying
D  first_name                    25a   varying
D  is_manager                    1a
```

   Assume that file **data.xml** contains the following lines, where "DBCS" represents double-byte data:

```
<DBCS_employee_info DBCS_is_manager="y">
   <DBCS_last_name>Smith</DBCS_last_name>
   <DBCS_first_name>John</DBCS_first_name>
</DBCS_employee_info>
```

   Option *case=convert* is specified. After the conversion of the non-alphanumeric characters to a single underscore, the name *DBCS_last_name* is converted to *_LAST_NAME*. Since RPG does not support names starting with an underscore, the initial underscore is removed. The final converted name is *LAST_NAME*.

```
xml-into employee_info %xml('data.xml'
                          : 'doc=file case=convert '
                          + 'ccsid=ucs2');
// employee_info.last_name = 'Smith'
// employee_info.first_name = 'John'
// employee_info.is_manager = 'y'
```