Source Scan Extensibility Exercise

Objective

The goal of this exercise is to familiarize users with the TPF Toolkit Source Scan extensibility tools. Upon completion, you should be able to perform the following tasks:

Exercise 1 – Using the source scan tool for C/C++

- Part A – Use the IBM provided template to create a custom C/C++ rule
- Part B – Create a fix for the error flagged by the rule
- Part C – Use a precondition to enhance the rule

Exercise 2 – Using the source scan tool for HLAsm

- Part A – Use the IBM provided template to create a custom HLAsm rule
- Part B – Create a fix for the error flagged by the rule
- Part C – Use a precondition to enhance the rule

Exercise 1 – Using the source scan tool for C/C++

Part A – Creating a custom C/C++ rule

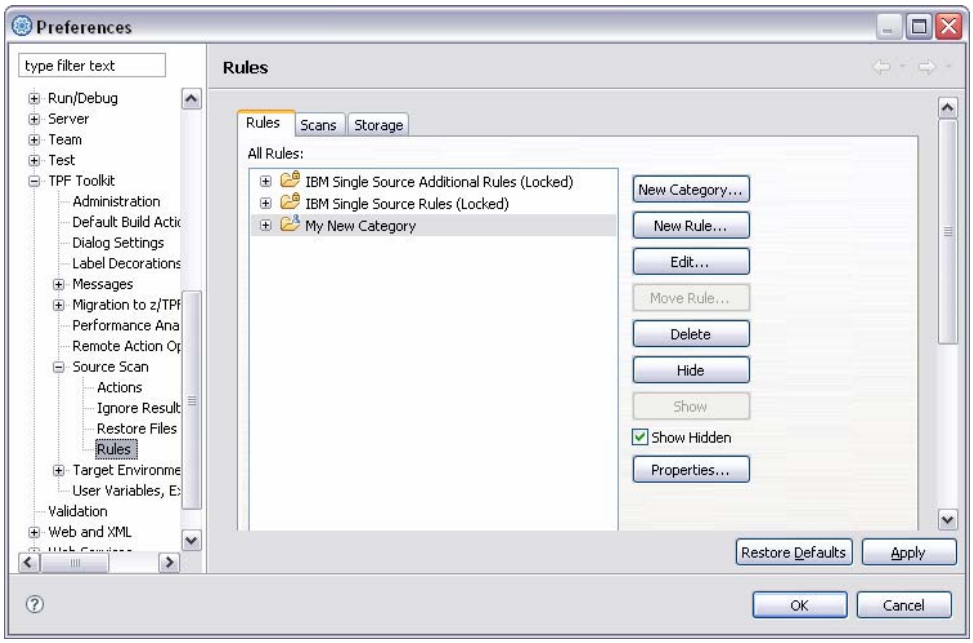
Purpose: Create a custom rule to flag all occurrences of the **`int`** type and replace it with **myInt**.

```
#include <stdlib.h>
#include <stdio.h>

struct myStruct {
    int structInt;
    double* structDouble;
};

int main() {

    int x;
    int y;
    int* p;

    /* we are ready to start */
    int i = 0;
    for (i = 0 ; i < 10 ; i++) {
         printf("%d ", i);
    }

    myStruct structVar;

    scanf("%d %d\n", &x, &y);

    return 0;
}
```

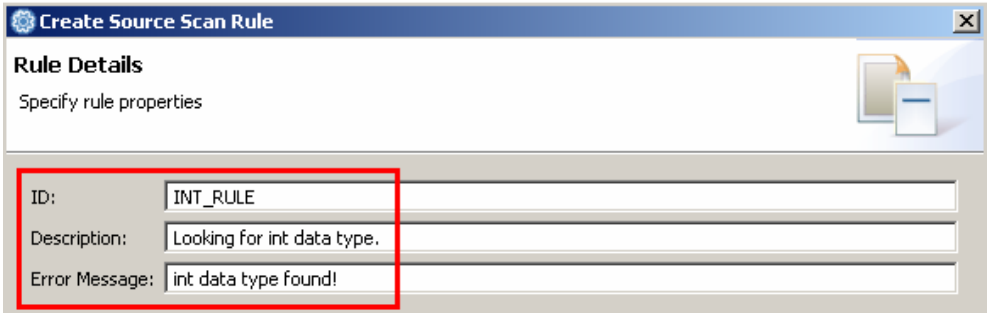
The outcome of this portion of the exercise will be a custom scan defined in TPF Toolkit which will be used to scan the above code snippet and flag occurrences of the `int` data type.

Actions:

1) Select Window → Preferences to bring up the Preferences dialog.
2) In the left pane, expand **TPF Toolkit** → **Source Scan**, and select **Rules**
3) The Preferences dialog should appear as shown below.



4) Click the **New Rule…** button.
5) Enter the following information into the **ID**, **Description** and **Error Message** fields:

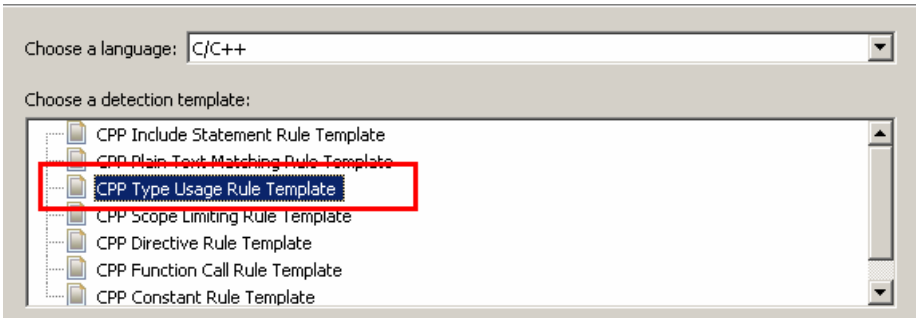


6) Optional. Inspect the other settings on this page:
   a. Does this rule flag a definite or potential problem?
   b. Does this rule flag an error or a warning?
   c. Which Icon do you want to mark the problem flagged by this rule in the Remote Error List?

   Note: For this exercise, leave the default selections for all the other settings on this page.

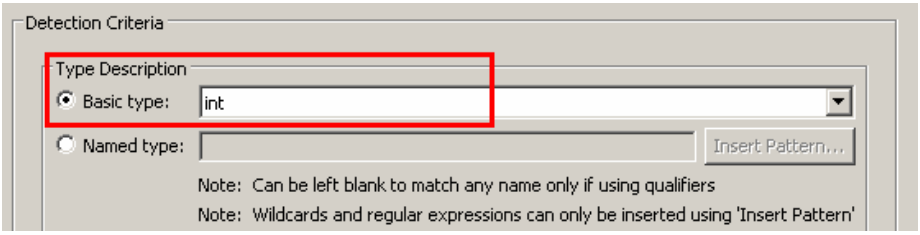
7) Click **Next** to continue.
8) On the **Select Rule Type** page, select the language that you are creating the rule for. Select **C/C++**
9) From the list of built-in templates, select the detection template that best describes what you want to detect.

   Tip: Since you are detecting the usage of **int**, which is a primitive type, select the **CPP Type Usage Rule Template**.



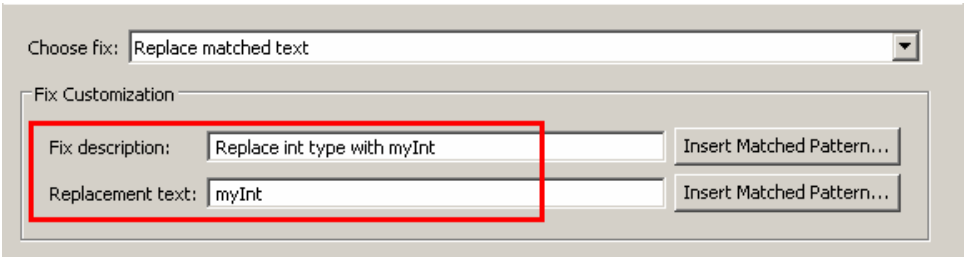
10) Click **Next** to continue to customize the detection template.
11) On the **Detection Template Customization** page, you can define the criteria which must be matched for a problem to be flagged by this rule.

    Tip: Since we are detecting the primitive type **int**, select the **Basic type** radio button and select **int** from the drop-down list:



12) Click **Next** to continue.
13) On the **Fix customization** page, you can specify how the problem flagged by this rule can be fixed.

    Tip: Since you are replacing **int** with **myInt**, select **Replace matched text** as the fix type. Then specify the following in the **Fix Customization** group:



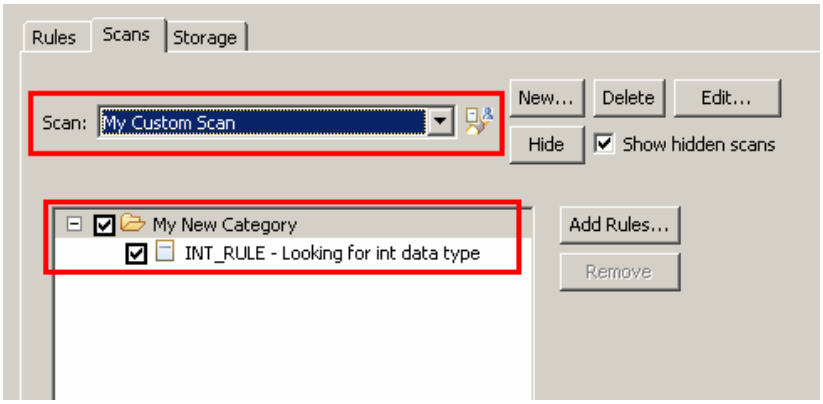
14) Click **Next** to continue.
15) Optional. The **Additional rule information** page allows you to specify how this rule will interact with other rules.

    Note: For this exercise, no changes are required on this page.

16) Click **Next** to continue.
17) On the **Parent Category** page, click the **Select** button and select the **My New Category** category.
18) Click **Finish**.

Congratulations! You have successfully created a custom C/C++ source scan rule.

Select the **Scans** tab and choose **My Custom Scan** from the drop-down list. You will see that the scan contains **My New Category** which in turn contains the new custom rule INT_RULE.



Part B – Scanning source files with your custom rule

In this section, we will be using the custom rule to scan C/C++ source files. A project called **Source Scan Demo** has already been created and is pre-populated with **exercise.c** that you can scan.

1) In the TPF Project Navigator, right-click on the **Source Scan Demo** TPF project and select **Source Scan → My Custom Scan**.
2) Notice that all **six** occurrences of the **`int`** data type have been flagged in the **exercise.c** file.
3) Apply the fix to the three occurrences of **int** outside of the struct
    a. Use the **Auto Correct**
        i. Select the error for line 11 in the **Remote Error List** view, right mouse click and select **Auto Correct Source Scan Problem**.
        ii. Click **Yes** in the confirmation dialog.
    b. Use **Quick Fix**
        i. Left mouse click on the error marker on line 13 and double-click on **Replace int type with myInt**.
        ii. Save the file.
    c. Use the **Compare Editor**
        i. Right mouse click an error in the **Remote Error List** and select **Compare File with Corrected Version**.
        ii. In the compare editor, use the  button to navigate to the last occurrence of int outside of the struct (line 14).
        iii. Click the  button to copy the suggested change to your file.

Congratulations! You have fixed three errors using three different fix mechanisms.

Part C – Adding a precondition to the rule

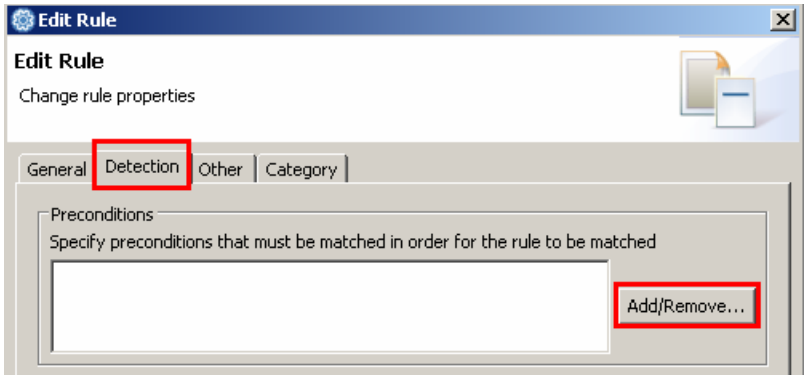
In this part of the exercise, we will add a *precondition* to the custom rule. A precondition on a rule is used to specify what additional conditions must occur in a source file before the rule can be flagged in the file.

Precondition: Require that the `int` data type appear within a `struct` definition.

Fix: When an int appears within a `struct`, change the type of the variable from `int` to `myInt_struct`

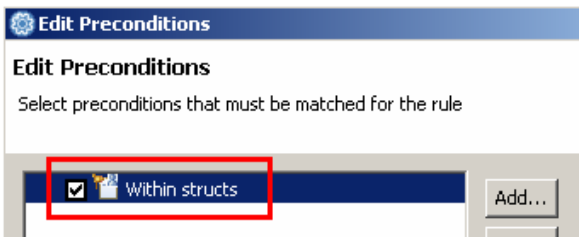
Actions:

1) Select Window → Preferences to bring up the Preferences dialog.
2) In the left pane, expand **TPF Toolkit** → **Source Scan**, and select **Rules**.
3) Expand the **My New Category** category and select the custom rule **INT_RULE**.
4) Click the **Edit…** button.
5) The **Edit Rule** dialog allows you to change various properties of the custom rule. For this exercise, select the **Detection** tab and click the **Add/Remove…** button in the **Preconditions** group:



6) In the **Edit Preconditions** dialog, click the **Add…** button to add a precondition.
7) Specify the details of the precondition as follows:
   a. Specify the **name** for the precondition.
   b. Specify the **location** where the precondition should occur relative to the error location. For this exercise, since we are detecting the error location (the usage of the `int` type) in the same statement as the precondition (`struct`), select **Same Line** from the drop-down list.
   c. Specify the detection criteria. For this exercise, we will use the **Scope Limiting Template** to specify how the precondition will be detected.
   d. Click the **Customize…** button for the **CPP Scope Limiting Rule Template**.
   e. In the **Precondition Detection Criteria** dialog, indicate that the error location has to be contained within a **Structure**.
   f. Click **OK** to save the precondition scope definition.
   g. Click **OK** once again to save the precondition definition.

8) Enable the precondition on the **Edit Preconditions** dialog by checking the box beside the new precondition



9) Click **OK** to save the changes.
10) To customize the fix, click the **Customize…** button beside the **Fix Template** drop-down list.
11) Change the **Replacement Text** to **`myInt_struct`**
12) Save the changes.
13) Switch to the **TPF Toolkit** perspective and clear the **Remote Error List** if there are errors from a previous scan.
14) In the **TPF Project Navigator**, right-click the **Source Scan Demo** TPF project and select **Source Scan → My Custom Scan**.
15) Notice that only the **`int`** within the **`struct`** is flagged by the rule INT_RULE.
16) Select the error in the error list. Right-click and invoke **Auto-correct Source Scan Problem**.
17) The **`int`** type within the `struct` will be replaced with **`myInt_struct`**:

```
4 struct myStruct {
5 /*     int structInt;*/
6      myInt_struct structInt;
7      double* structDouble;
8 };
```

Exercise 2 – Using the source scan tool for HLAsm

Part A – Creating a custom HLAsm rule

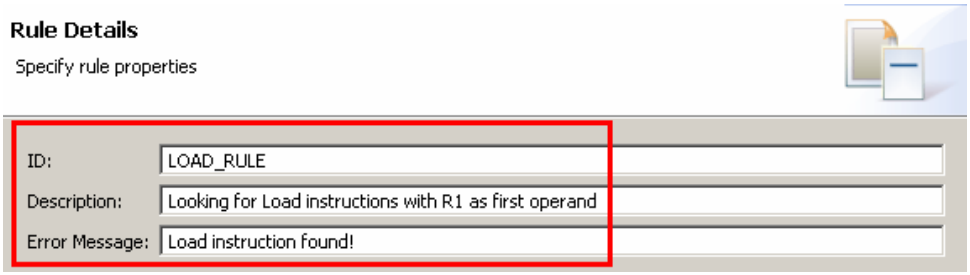
Purpose: Create a custom rule to flag all occurrences of the **LOAD** (L) instruction that has R1 as the first operand. The fix for this rule will replace the R1 operand with R14.

```
BEGIN NAME=TEST

L       R1,CE1SVD
L       R1,CE1S00
L       R1,CE1S04

LCPCC PROTECT='Some long string',                              X
       CR0SAVE='Some other long string'

L       R1,CE1S05
LM      R1,R7,CE1SVR
STM     R1,R2,CE1S01
ST      R2,CE1S03
ST      R3,CE1S04
L       R3,CE1S00

LCPCC PROTECT='Some long string',                              X
       CR0SAVE='Some other long string'

LCPCC PROTECT='Some long string                                X
       rest of string',CR0SAVE=hello

ST      R4,CE1S05
L       R1,CE1S00

FINIS TEST
END
```

The outcome of this portion of exercise will be a custom scan defined in TPF Toolkit which will be used to scan the above code snippet and flag the Load instructions.

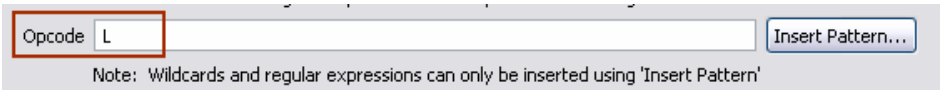
Actions:

1) Select Window → Preferences to bring up the Preferences dialog.
2) In the left pane, expand **TPF Toolkit** → **Source Scan**, and select **Rules**.
3) Click on **New Rule…** and enter the following information into the **Rule Details** page:

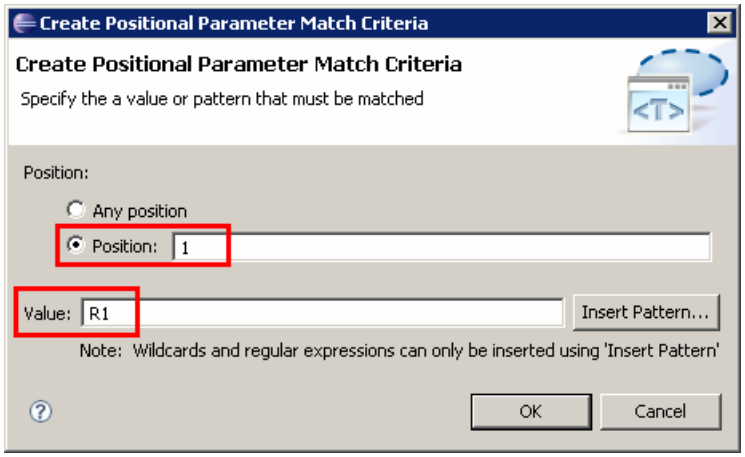


4) Click **Next** to go to the **Select Rule Type** page.
5) Select **HLAsm** as the language type and click **Next**.

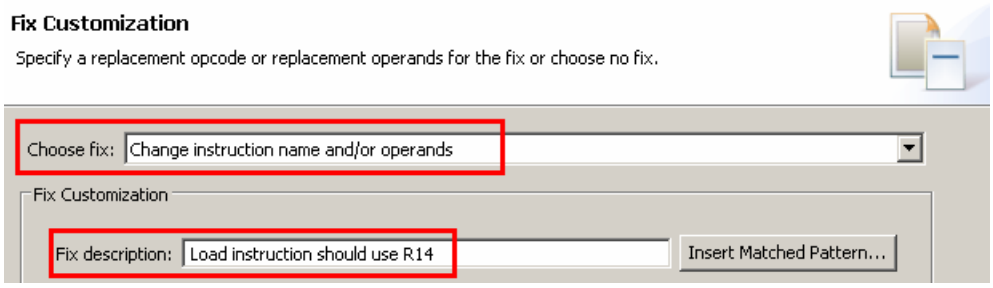
6) Specify that you want to look for the Load (L) instruction.



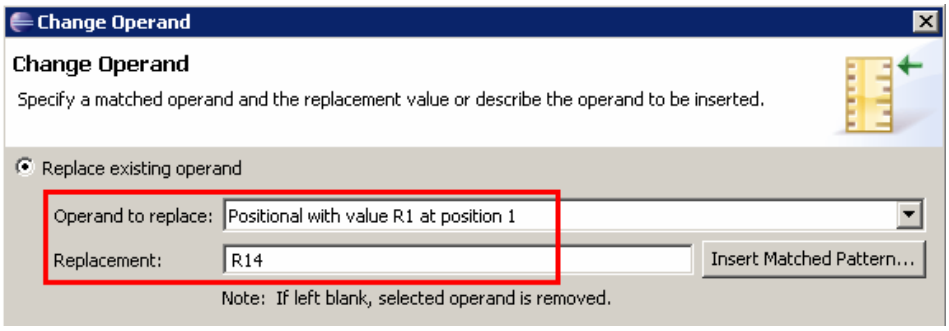
7) Click the **Add…** button in the **Positional Operands** group and specify that you want to look for **R1** as the first operand.



8) Click **OK** to add this to the list.
9) Click **Next** to move to the **Fix Customization** page.
10) Select the **Change instruction name and/or operands** fix. Enter a description for the fix.



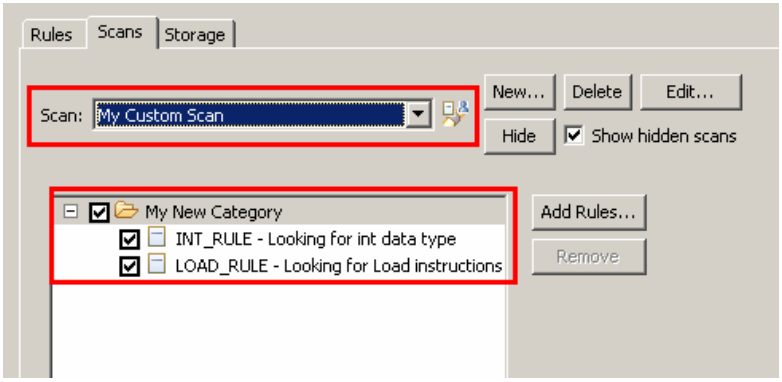
11) In the **Change Operands** group, click the **Add…** button.
12) In the **Change Operand** dialog, specify that you want to replace the detected R1 operand at position 1 with R14:



13) Click **OK** to save the changes.
14) Click **Next** twice to advance to the **Parent Category** page.
15) Select **My New Category** and click **Finish**.

Congratulations! You have successfully created a custom HLAsm source scan rule.

Select the **Scans** tab and choose **My Custom Scan** from the drop-down list. You will see that the scan contains **My New Category** which in turn contains the new custom rule LOAD_RULE.



Part B – Scanning source files with your custom rule

In this section, we will be using the custom rule to scan ASM source files. A project called **Source Scan Demo** has already been created and is pre-populated with **exercise.asm** that you can scan.

1) In the **TPF Project Navigator**, right-click on the **Source Scan Demo** TPF project and select **Source Scan → My Custom Scan**.
2) Notice that five lines with the Load instruction are flagged in **exercise.asm**.
3) Apply the fix to the first three occurrences of the load instruction
    a. Use the **Auto Correct**
        i. Select the error for line 3 in the **Remote Error List** view, right mouse click and select **Auto Correct Source Scan Problem**.
        ii. Click **Yes** in the confirmation dialog.
    b. Use **Quick Fix**
        i. Left mouse click on the error marker on line 5 and double-click on **Load instruction should use R14**.
        ii. Save the changes.
    c. Use the **Compare Editor**
        i. Right mouse click an error in the **Remote Error List** and select **Compare File with Corrected Version**.
        ii. In the compare editor, click the  button to copy the first suggested change to your file.
        iii. Save the file.

Congratulations! You have fixed three errors using three different fix mechanisms.

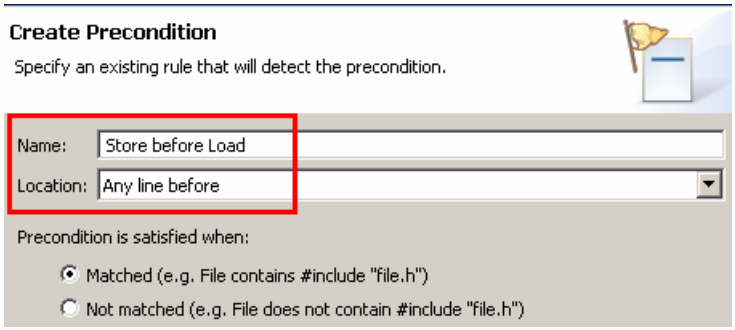
Part C – Adding a precondition to the rule

In this part of the exercise, we will add a *precondition* to the custom rule. A precondition on a rule is used to specify what additional conditions must occur in a source file before the rule can be flagged in the file.

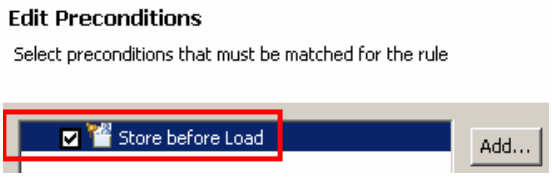
Precondition: Require that Load instructions with R1 as the first operand must appear after a Store (ST) instruction.

Actions:

1) Select Window → Preferences to bring up the Preferences dialog.
2) In the left pane, expand **TPF Toolkit** → **Source Scan**, and select **Rules**.
3) Expand the **My New Category** category and select **LOAD_RULE**.
4) Click the **Edit…** button to add the precondition.
5) Click the **Add/Remove…** button in the **Preconditions** group.
6) Click **Add…** and specify the following information:
    a. Specify a **name** to help you identify the precondition.
    b. For the precondition **location**, specify that the precondition (ST instruction) can occur anywhere before the error location (L instruction).



7) In the **Detection Criteria** group, select the **New Condition** radio button to indicate that you want to customize the precondition detection criteria.
8) Click the **Customize…** button to open the **Precondition Detection Criteria** dialog.
9) Specify **ST** as the **Opcode** that the precondition should detect.
10) Click **OK** to save the changes for the precondition detection criteria.
11) Click **OK** again to save the precondition definition.
12) Select the **Store before Load** precondition that you just created and click **OK**:



13) Click **OK** on all the dialogs to save the changes.
14) In the **TPF Project Navigator**, right-click on the **Source Scan Demo** TPF project and select **Source Scan** → **My Custom Scan**.
15) Notice that only the Load instruction on line 14 is flagged.