

Pipelines

This chapter describes how to create, test, and start and stop pipelines. Pipelines are groups of stages. It also describes how effective dates determine which version of a configuration file the Pipeline Server uses to process sessions.



See Also:

- For a conceptual description of pipelines, see “Important Concepts” in *Getting Started*.
- For information on resolving problems with pipelines, see *Troubleshooting*.

Service Definition

All sessions metered to the Pipeline server belong to a service. A service is a type of task that can be performed by a service element for a client. For example, rating a telephone call. A service definition is a configuration file that defines all the properties that comprise a service. These properties are processed by plug-ins. A service definition follows the MSIX protocol and standards for naming conventions and defining types.

Metering errors, such as missing properties detected based on the service definition, are captured and reported.

Creating the Service Definition

This is Step 2 of the Platform Manager Web Service Creation Wizard. See the Platform Manager online help or the *Platform Extensions Guide* for details.

If you are not using the Wizard, then you manually do the following:

1. Create a service definition file (*service_name.msixdef* where *service_name* is the name of the service). See “MSIX Service Definition File” on page 23 for the layout of a service definition file.
2. Create the AutoSDK file for the service definition. (A sample path is *installation\test\autosdk\test_service_name.xml* where *installation* is the directory containing the MetraTech software and *service_name* is the name of the service.)



See Also

For details, see *Error Handling and Testing*.

MSIX Service Definition File

The MSIX service definition file contains MSIX-specific definitions for the service. The tags conform to the MSIX protocol.

Descriptions of the properties in the MSIX service definition follow:

- **dn:** Distinguished name. This is the actual property name.
- **type:** Data type. Valid values are int32, string, unistring, timestamp, float, double, boolean, enum and decimal.

- **length:** Maximum length in characters (not to exceed 255) for the value of properties with a string data type. This property is reserved for future use.
- **required:** Whether a value is required. Valid values are **Y** (a value is required—a NOT NULL column in the database table) or **N** (a value is not required—a NULL column in the database table).
- **defaultvalue:** The default values for the property in the database column. If a property is not required, and no value is metered in, the the default value will be used during processing.

A sample MSIX service definition from the testservice.msixdef file follows:

```
<defineservice>
  <name>metratech.com/testservice</name>
  <description>Simple test service. Used for testing basic
  functionality and SDK connectivity.</description>

  <ptype>
    <dn>description</dn>
    <type>string</type>
    <length>80</length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
    <description>Comment entered by the user. Used for
    display only.</description>
  </ptype>
  <ptype>
    <dn>time</dn>
    <type>timestamp</type>
    <length></length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
    <description>Any date/time value. Used for
    display only.</description>
  </ptype>
  <ptype>
    <dn>units</dn>
    <type>float</type>
    <length></length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
    <description>Arbitrary number of units that will be rated
    for. The units are multiplied by a configurable rate to
    calculate the amount.</description>
  </ptype>
  <ptype>
    <dn>accountname</dn>
    <type>string</type>
    <length>20</length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
    <description>Account to which this transaction is
    associated. Used for account resolution.</description>
  </ptype>
  <ptype>
    <dn>DecProp1</dn>
    <type>DECIMAL</type>
    <length></length>
```

```

        <required>N</required>
        <defaultvalue>0.000000</defaultvalue>
        <description>Test decimal property</description>
    </ptype>
    <ptype>
        <dn>DecProp2</dn>
        <type>DECIMAL</type>
        <length></length>
        <required>N</required>
        <defaultvalue>0.000000</defaultvalue>
        <description>Test decimal property</description>
    </ptype>
    <ptype>
        <dn>DecProp3</dn>
        <type>DECIMAL</type>
        <length></length>
        <required>N</required>
        <defaultvalue></defaultvalue>
        <description>Test decimal property</description>
    </ptype>
</defineservice>

```

Product View Definition

The product view definition specifies properties that can be displayed on the MetraTech Presentation Server and which properties created by the Pipeline server are stored. The data in the product view is the result of the processing that occurred while the stages were running. The product view definition follows the MSIX protocol and standards for naming conventions and defining types.

Physical tables in the database are created using configuration files for product view definitions. Every property in the MSIX product view definition file maps to a column in the database table that gets created. Data in the table appears to the user as details of the interactive bill.

The prefix **t_pv** in the following example indicates a table product view:

```

t_pv_metrattech_com_testservice_1000
t_pv_metrattech_com_testservice_1001
t_pv_metrattech_com_testservice_1002

```

Location

You create a product definition file (*productview_name.msix* where *productview_name* is the name of the product view) in the following area:

```
install\dir\Extensions\extension_name\config\productview\site_name\
```

where:

- *install\dir* is where you installed the MetraTech Platform software,
- *extension_name* is the name of the Platform Extension that uses the product view, and
- *site_name* is the name of the site where subscribers will log in to view their bills.

For example, one of the product view definition files supplied with the audio conferencing platform extension has the following path (assuming you installed the MetraTech software in the default folder):

```
C:\MetraTech\RMP\rmp\Extensions\audioconf\config\productview\metratech.com\
audioConfCall.msixdef
```

The MSIX definition file contains MSIX-specific definitions for the product view. The tags conform to the MSIX protocol.

Properties

Descriptions of the properties in the MSIX product view definition file follow. Be aware that the product view definition cannot contain reserved fields (indicated by an underscore). For example, do not include `_Amount` or `_AccountID`.

- **ptype:** this tag is used to begin and end each property. It contains several useful attributes: *exportable*, *filterable*, and *uservisable*.
- **dn:** Distinguished name. This is the actual property name.
- **type:** Data type. Valid values are `int32`, `string`, `unistring`, `timestamp`, `float`, `double` and `decimal`.
- **length:** Maximum length in bytes for the value of properties with a string data type. This determines the column width in the database.
- **required:** Whether a value is required. Valid values are **Y** (a value is required—a NOT NULL column in the database table) or **N** (a value is not required—a NULL column in the database table). This property is not currently supported.
- **defaultvalue:** The default values for the property in the database column. This property is not currently supported.

Attributes

Product views support the following optional attributes:

- **exportable:** in MetraTech Presentation Server (MPS), determine whether a property should be exported to a CSV (comma separated file) when the subscriber selects the **Export to CSV** feature from within MPS
- **filterable:** determines whether a property should be shown in the property list dropdown used on the filter screen in MPS.
- **uservisible:** determines whether the property should be visible (that is, returned from the database) when data is queried. Setting this to `False` essentially sets the previous two attributes to `false`, as well.

All of these attributes are true by default; product view properties are user visible, exportable and filterable by default. If you want different behavior, you must explicitly set the attributes to `False` within the product view configuration file.

Let's take a look at a few examples of how to use these attributes.

If you wish to have a property that is persisted in the database but should not be displayed to the subscriber, mark it as not uservisible. An example is any property that is used internally

during processing, but has no relevance to the subscriber viewing their bill from within the MetraTech Presentation Server application. The property would look like the following:

```
<ptype uservisible="false">
  <dn>InternalGeneralLedgerCode</dn>
  <type>unistring</type>
  <length>30</length>
  <required>Y</required>
  <defaultvalue></defaultvalue>
</ptype>
```

If you wish to have a property that is used in MPS for rendering the page but not exported or listed as a filter property, mark it as not exportable and not filterable. An example of this might be a URL that was metered to the system. When we display the item, we may create an HTML link using the information that is returned, but do not want the subscriber to filter on this data or possibly even to export it. The property would look like the following:

```
<ptype exportable="false" filterable="false">
  <dn>ThirdPartyItemDescriptionURL</dn>
  <type>unistring</type>
  <length>100</length>
  <required>N</required>
  <defaultvalue></defaultvalue>
</ptype>
```

Example

A sample MSIX product view definition from the recurringcharge.msixdef file follows:

```
<defineservice>
  <name>metratech.com/recurringcharge</name>
  <!-- <![CDATA[ ]]> -->
  <description>
    <![CDATA[
      RecurringCharge. Metered by the recurring charge usage
      server adapter at the end of the billing period. The
      recurring charge amount is determined in the pipeline
      and can not be metered.
    ]]>
  </description>

  <ptype>
    <dn>_Intervalid</dn>
    <type>int32</type>
    <length></length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
    <description>The ID of the Usage Server
      Interval</description>
  </ptype>

  <ptype>
    <dn>intervalstartdate</dn>
    <type>timestamp</type>
    <length></length>
    <required>Y</required>
    <defaultvalue></defaultvalue>
```

```

        <description>The start date of the current
        interval.</description>
    </ptype>

    <ptype>
        <dn>intervalenddate</dn>
        <type>timestamp</type>
        <length></length>
        <required>Y</required>
        <defaultvalue></defaultvalue>
        <description>The end date of the current interval
        </description>
    </ptype>

</defineservice>

```

Optional Properties

Properties in service definitions and product view definitions can be made optional by putting a value of **N** in the `<required>` tag. Here is an example of an optional property:

```

<ptype>
    <dn>email</dn>
    <type>string</type>
    <length>100</length>
    <required>N</required>
    <defaultvalue></defaultvalue>
</ptype>

```

By setting the `<required>` tag to **N**, this property is no longer required in the session. This means that if this property is not metered, the session will not fail.

An optional property can have a default value. A default value is a value assigned to the property if it was not metered in (in the case of a service definition) or found in the session (in the case of a product view definition). For example:

```

<ptype>
    <dn>email</dn>
    <type>string</type>
    <length>100</length>
    <required>N</required>
    <defaultvalue>jobs@metratech.com</defaultvalue>
</ptype>

```

If this property was part of a service definition and no value was metered in, then this property would be set in the current session with the value of **jobs@metratech.com**. If the `<defaultvalue>` tag had no value, then the property would simply not exist in the session.

If this property was part of a product view definition and no value for it was found in the session, then this property would be written to the database as **jobs@metratech.com**. If the `<defaultvalue>` tag had no value, then the property would be written as a NULL in the database. One exception to this case is if the property was an enumerated type; then it would be written to the database as **0** to ensure proper localization later.

Certain combinations of optional properties in service definitions and their respective product views can be very useful. For example, we can have a property that is optional or even absent in a service definition but required in the product view. This means that by the time the

session arrives at the WriteProductView stage, the property must have been put in the session, either originally by metering, or by some plug-in setting the property in the session.

You could even have two different sets of default values (one for the service definition and one for the product view) for an optional property. This provides a great deal of flexibility for solving many different design issues.

Enumerated Types

You can set up enumerated type definitions for your services. For an example, see “Enumerated Types and Localized Strings” on page 15.

Using enumerated type definitions eliminates the requirement for storing strings in the database and lets you localize your service precisely and efficiently.

The pipelines use these enumerated values to map metered enumerated values to the localized strings that the MetraTech Presentation Server displays.

Using the Platform Manager application, you can perform the following enumerated type configuration tasks:

- **Add a new *enumspace*.** An enumspace is an umbrella (typically your service domain name) which holds an enumtype or set of enumtypes. Within each enumspace, all enumtypes and their values must be unique; across enumspaces, however, enumtypes can be the same. A sample enumspace might be "myservice.com."
- **Add a new *enumtype*.** An enumtype is a user-defined type consisting of a set of named constants called enumerators. A sample enumtype might be "service level."
- **Add a new *enumerator*.** An enumerator is a constant that gets assigned a value which is metered to the pipeline. A sample enumerator might be "gold."
- **Add a new *localized string*.** A localized string is the language for your enumerator.
- **Add a new *value*.** The value is the description ID which gets metered through the pipeline. You can have multiple values per enumerator. This value can be an integer or a string. For details, see the **Enumerator Name/Value Table** on page 17.

See The Platform Manager online help for details.

Configuring the Enumerated Type Definition

You should use Platform Manager to configure all enumerated types. However, if you want to view the underlying files, this section presents an example.

Enumerated Type files reside in folder hierarchy for the Platform Extension they are associated with. For example, the audio conferencing extension has enumerated type files in the following folder:

```
installdir\Extensions\audioconf\config\enumtype\metratech.com
```

where *installdir* is the directory containing the MetraTech software.

Enumerated Type Configuration File Format

Definitions of the tags in an enumerated type definition files follow:

- **enum_spaces:** Each service can have its own enum space, though normally groups of services will share a single enum space.
- **enum_space:** each enum space must have a name property. The name corresponds to a path and file name for the ...
- **description:** contains a text description for the enum space.
- **enums:** this tag indicates that enumerated types follow
- **enum:** each enum tag must have a name property; this is the identifier for the enumerated type.
- **description:** contains a text description for the enumerated type.
- **entries:** contains a list of name/value pairs for the enumerators in the enumerated type.
- **entry:** contains a single value for the enumerator. The name property typically matches the <value> tag, as in the following example:

```
<entry name="Monday">
  <value>Monday</value>
</entry>
```



Note

We recommend that you not edit these files by hand; use the Platform Manager application to add and edit enumerated types. When you edit the files manually, it is far too easy to introduce an error and corrupt the XML file.

See the Platform Manager online help for more information.

Here is the organization for an enumerated type configuration file:

```
<mt_config>
  <enum_spaces>
    <enum_space name="metratech.com/audioconfconnection">
      <description>Metratech namespace</description>

      <enums>
        <enum name="CalendarCode">
          <description>Calendar Settings</description>
          <entries>
            <!-- list of entry name/value pairs -->
          </entries>
        </enum>

        <!-- next enumerated type: <enum name></enum name> -->

      </enums>
    </enum_space>
  </enum_spaces>
</mt_config>
```


Example

Let's take a look at an example of how enumerated types are defined. If you examine the Audio Conferencing extension in Platform Manager, you'll see that there are three namespaces:

- "metratech.com/audioconfconnection"
- "metratech.com/audioconfconnection"
- "metratech.com/audioconfPlayBack"

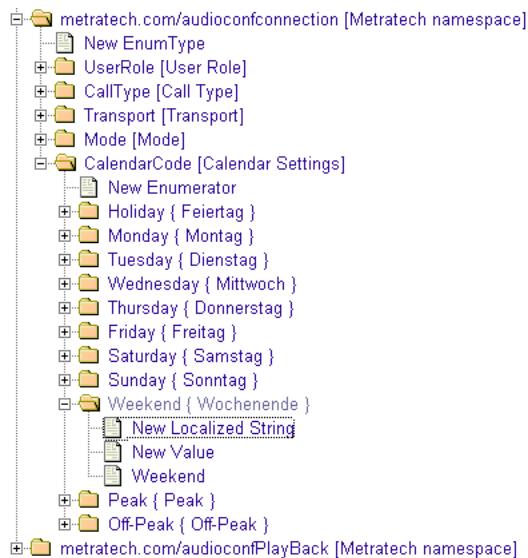
And you'll see that the **audioconfconnection** namespace has several EnumTypes:

- UserRole
- CallType
- Transport
- Mode
- CalendarCode

If you examine one of these, say **CalendarCode**, you'll see the available enumerators: Holiday

- | | |
|-------------|------------|
| ■ Monday | ■ Saturday |
| ■ Tuesday | ■ Sunday |
| ■ Wednesday | ■ Weekend |
| ■ Thursday | ■ Peak |
| ■ Friday | ■ Off-Peak |

And then there are values and Localized strings that appear depending on which country you're viewing. You see the entry name field, then the value next to it in brackets. For example, this figure shows the German enumeration values:



Localization Country:

Figure 10 Enum Space Example

If you choose a different language, you'll see a different value for each enumerator entry name.