MTP Device Simulator User’s Guide

November 5, 2008

Abstract

This document describes the Media Transfer Protocol (MTP) Device Simulator—MtpSim—and its capabilities. The tool assists product development by enabling developers to parse MTP command packets and analyze an MTP device’s response based on the MTP specification.

This information applies for the following operating systems:  
 Windows® 7  
 Windows Vista®

References and resources discussed here are listed at the end of this paper.

For the latest information, see:   
 <http://www.microsoft.com/whdc/>device/wpd/MTPTS\_Guide.mspx

Disclaimer: This is a preliminary document and may be changed substantially prior to final commercial release of the software described herein.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, Windows, Windows Media, Windows Vista, and Xbox are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| November 5, 2008 | First publication | | | |

Contents

[Introduction 4](#_Toc212541823)

[MTP Basics 4](#_Toc212541824)

[MtpSim Supported Features 4](#_Toc212541825)

[Transports 5](#_Toc212541826)

[SoftUSB 5](#_Toc212541827)

[Named Pipes 5](#_Toc212541828)

[Bluetooth 5](#_Toc212541829)

[Netchip (USB) 6](#_Toc212541830)

[IP 6](#_Toc212541831)

[Legacy MTP 6](#_Toc212541832)

[MTP Device Services for Windows 7 6](#_Toc212541833)

[Platform Compatibility 7](#_Toc212541834)

[Windows XP x86 7](#_Toc212541835)

[Windows XP x64 7](#_Toc212541836)

[Windows Vista x86 7](#_Toc212541837)

[Windows Vista x64 7](#_Toc212541838)

[Windows 7 x86 and x64 8](#_Toc212541839)

[Using MTP Device Simulator 8](#_Toc212541840)

[Installation 8](#_Toc212541841)

[Command-Line Options 8](#_Toc212541842)

[User Interface 9](#_Toc212541843)

[Troubleshooting 10](#_Toc212541844)

[Device and Service Capability Configuration 10](#_Toc212541845)

[Device Stage Configuration 10](#_Toc212541846)

[Default Device Stage Configuration 10](#_Toc212541847)

[Enhanced Device Stage Configuration 10](#_Toc212541848)

[Replacing Device Stage with Third-Party Profiles 12](#_Toc212541849)

[Plug-in Mechanism 14](#_Toc212541850)

[Loading Plug-ins 14](#_Toc212541851)

[Unloading Plug-ins 14](#_Toc212541852)

[MtpSim Built-in Plug-ins 14](#_Toc212541853)

[MTP Trace Plug-in (optrace.dll) 15](#_Toc212541854)

[Fault Injection Plug-in (injfault.dll) 16](#_Toc212541855)

[Browser Plug-in (objbrowser.dll) 19](#_Toc212541856)

[Resources 24](#_Toc212541857)

[Appendix A. Windows 7 MTP Device Services Extensions 25](#_Toc212541858)

[Appendix B. Troubleshooting 26](#_Toc212541859)

[Appendix C. Command-Line MtpSim Commands 27](#_Toc212541862)

[Appendix D. Run-Time MtpSim Commands 28](#_Toc212541863)

[Appendix E. INI File Configuration 29](#_Toc212541864)

[Appendix F. XML Configuration File Details 31](#_Toc212541870)

# Introduction

Media Transfer Protocol (MTP) Device Simulator—MtpSim— is a tool that helps developers during product development. This document describes the tool’s architecture and provides information on the simulator’s capabilities and configuration options. MtpSim enables developers to parse MTP command packets while exercising expected behavior based on the MTP specification, by returning appropriate MTP response packets and generating MTP events where required. The tool’s graphical user interface (GUI) enables easy configuration of MTP parameters and data presentation. The simulator also provides the framework for custom plug-ins to extend the tool’s functionality.

# MTP Basics

MTP is a protocol for transferring audio, video, or image content from a host PC to a portable media player, digital camera, personal digital assistant (PDA), or other portable device. Microsoft developed MTP as an extension of the Picture Transfer Protocol (PTP) ISO 15740 standard. PTP was developed to provide a standard method by which digital signal cameras (DSCs) can transfer image files and metadata to and from a PC. Although PTP was designed specifically for transferring picture data, MTP was designed to be a more general media solution. Because MTP is an extension of PTP, all PTP protocol semantics are valid in MTP. In the Windows® 7 operating system, MTP Device Services Extensions were added to deal with a broader set of scenarios that are specific to mobile phone connectivity with the PC. For details, see Appendix A and “MTP Device Services Extensions” in the “Resources” section of this document.

MTP uses asynchronous communication that is based on a command and response structure. Two devices are involved in MTP data transfer—the initiator and the responder. The PC typically is the initiator, whereas the media device is always the responder. A non-PC device can also be an initiator (such as a car stereo head unit or a Microsoft® Xbox® 360). A typical responder is a portable media player that plays audio files or DSCs, or a mobile phone that can take photographs. A sophisticated responder might have additional capabilities—for example, a mobile phone may be able to capture video and play media files. The responder connects to an initiator that supports transferring audio, video, and image files to and from the responder. Windows Media® Player is an example of an MTP initiator that is supported on a PC. The responder communicates with the initiator through a USB, Bluetooth, or network connection (typically, wireless). For the scope of this document, the initiator is the Windows 7 operating system and the responder is the MTP Device Simulator.

# MtpSim Supported Features

MtpSim supports all MTP device capabilities, object properties, and operations that are based on the [MTP Specification](#_Supplemental_Information:). The tool also supports multiple transport options such as USB, SoftUSB, Bluetooth, and IP. The tool runs on the Windows Vista® and Windows 7 operating systems.

## Transports

### SoftUSB

MtpSim uses the Device Simulation Framework (DSF) to create a virtual USB device that "plugs in" to your system. DSF must be installed on the test PC before connecting in SoftUSB mode. DSF is installed as part of the MtpSim installation. You can start the simulator in SoftUSB mode by specifying the /d parameter. The simulator can be run in SoftUSB mode on Windows XP, Windows Vista, and Windows 7.

### Named Pipes

You can use the Named Pipes mode to send data from the MtpSim tool directly to the MTP driver, bypassing all transport drivers.

### Bluetooth

Deploying MtpSim over a Bluetooth interface (MTP/BT) requires a two-computer configuration (system A and system B). Here the initiator (System A) is a PC that is running Windows 7 and the other PC (System B) is running MtpSim as the MTP/BT responder. The system that is running MtpSim can be running either Windows 7 or Windows Vista. A Bluetooth dongle is required on the test system to run MtpSim in Bluetooth transport mode. Also, the Bluetooth MTP responder driver must be installed manually by running the command **InstSwMtpDev.exe /i**, which can be found under the c:\mtpsim\BthMtpResp directory.

To run MtpSim on Bluetooth transport

To connect and pair with Bluetooth:

1. On System B, double-click the **Bluetooth device** icon on the taskbar, open the context menu, select **Bluetooth settings**, and then select the **Allow Bluetooth device to find this computer** check box.

2. On System A, double-click the **Bluetooth device** icon on the taskbar, and then click **Add Bluetooth device**. Select **System B**, and then click **Add**.

A passcode is created in System A.

3. On System B, a dialog box appears with a request for pairing. Enter the passcode from step 2.

4. On System B, run **MTPSimUi.exe –b** to start MTPSIM in Bluetooth mode.

To connect after pairing with Bluetooth:

1. On System B, start MtpSim in Bluetooth mode, under Transport, select **Bluetooth**, and then under **Connection**, click **Connect** from the left panel in MtpSim.

2. On system A, from **Device Center**, double-click **System B**, and then select **Connect** or **Automatically Reconnect**.

Note  Do not run MtpSim as an MTP/BT responder on any non-test systems because this can interfere with regular Windows Bluetooth functionality.

### Netchip (USB)

This mode uses a Netchip 2280 USB Device card (NET2280-EVB from [www.PLXTECH.com](http://www.PLXTECH.com)) to create a physical USB MTP device. Then you can connect MtpSim to the same computer or another computer that is based on the configuration of your choice. For MtpSim to run in USB mode, the Netchip PCI card must be present on that PC and the Netchip drivers must be installed. You can find the drivers in the Device Enabling Kit.

### IP

The IP mode uses UPnP to report MtpSim as an MTP over IP (MTP/IP) device on a local network. You can connect MtpSim to the same computer or another computer that is based on the configuration of your choice. For MtpSim to run in IP mode, the IPv6 stack must be present. This is preinstalled on Windows Vista computers, but must be installed on Windows XP by using the ipv6 installation command.

When MtpSim is run in MTP/IP mode, Windows Firewall presents a dialog box that asks the user to unblock/block the port that MtpSim uses. For MtpSim to work correctly in IP mode, you must select the **Unblock** option. If the Windows Firewall prompt does not appear, make sure that an exclusion rule is set for MtpSim through the Firewall Control Panel program (firewall.cpl*)*.

Note  Two-computer configurations may have some difficulties with firewalls. For example, the Wireless Portable Devices port in Windows Firewall must be opened to enable MTP/IP communication.

## Legacy MTP

MtpSim supports all MTP device capabilities, object properties, and operations as specified in the MTP 1.0 Specification.

## MTP Device Services for Windows 7

MtpSim supports the following MTP Device Services by using the default configuration:

* Contacts
* Calendar
* Task
* Notes
* Ringtone
* Hints
* Status
* Device Metadata
* Full Enumeration Synchronization
* Anchor Synchronization

The other profiles that are provided with the MTP Device Simulator support device-specific services, formats, events, and methods that are defined for that device class. For additional information on these services, refer to Windows Portable Devices (WPD) Device Service Definitions in the “Resources” section at the end of this document.

# Platform Compatibility

The MtpSim simulator is supported on Windows XP with Windows Media Player 10, Windows XP with Windows Media Player 11, and Windows Vista platforms in USB mode. In named-pipe or soft mode, the simulator is supported on Windows XP with Windows Media Player 11 and on Windows Vista platforms. You can run the tool in soft mode on Windows XP with Windows Media Player 10 but will have limited support. MTP/IP mode is supported only on Windows Vista platforms. The SoftUSB mode on DSF is supported on all three platforms.

Note  On Windows Vista and Windows 7, MtpSim must be installed and used in an elevated command window or must be started in elevated mode. This is because administrator permissions are required to connect to the NetChip USB card or to emulate a PnP plug-in on the PipeMtp bus enumerator.

## Windows XP x86

MtpSim installation requires Windows Media Player 11+ to be already installed on the Windows XP computer. There is no additional requirement for installing the simulator on Windows XP x86.

## Windows XP x64

MtpSim has a dependency on wmvcore.dll, which is used to extract or update metadata in media files. Windows XP x64 does not include a 64-bit version of wmvcore.dll. Therefore, the 32-bit version of the MtpSim binaries (device-side only—Mtpsim.exe, MtpSimCr.dll, MtpSimNp.dll, MtpSimIP.dll, and MtpSimUS.dll) must be installed on a Windows XP x64 computer, and the simulator must run in Windows 32-bit on Windows 64-bit (WoW64) mode. You should use the 64-bit versions of the drivers because the MTP driver stack is native 64 bit. The installation script handles this and copies the appropriate 32-bit device-side MtpSim binaries.

## Windows Vista x86

To work correctly, MtpSim must run in an elevated mode. There are no additional requirements.

## Windows Vista x64

To work correctly, MtpSim must run in an elevated mode. Also, Windows Vista x64 does not trust test-signed kernel-mode drivers even when the test certificate is installed. To resolve this issue, "Test Mode" must be enabled on Windows Vista x64. You can do this by running the following command from an elevated prompt:

bcdedit.exe –set loadoptions DDISABLE\_INTEGRITY\_CHECKS

## Windows 7 x86 and x64

MTP Device Simulator is compatible with the Windows 7 Beta build.

The following are two known issues:

* The phone's default metadata package supports the media synchronization task but not the personal information manager (PIM) synchronization task.
* If you browse the device and select the storage by using the phone profile, you encounter a status task error.

# Using MTP Device Simulator

## Installation

To install MTP Device Simulator (MtpSim)

1. Unpack the MtpSim.cab to a directory such as c:\mtpsiminstall.

2. Start a Command Prompt window in elevated mode (cd c:\mtpsiminstall, run install.cmd).

3. Acknowledge the DSF Microsoft Software License Terms dialog box, choose the typical installation button, and then continue.

A "Launch MtpSim" icon is created on your desktop.

## Command-Line Options

You can control MtpSim at startup by using command-line parameters. This can be useful when you must set up test automation. You can change most of these parameters at run time as well. For details, see Appendix C. You can control the simulator at run time by using the keyboard if you start the simulator by using MtpSim.exe (the console handler). To view the tool’s available options at any time, depress the Enter key at the console. All command options in Appendix D are also configurable in MtpSimUI.exe, except for option ”C.”

Running mtpsimui.exe /? displays the available command-line options, as shown in the following screen shot.



Note  If no parameters are specified, the default mode is **Normal Mode** and the default connection type is SoftUSB with the default serial number set as 1.

## User Interface

The following screen shot shows the MtpSim user interface (UI) with navigation buttons and configuration options.

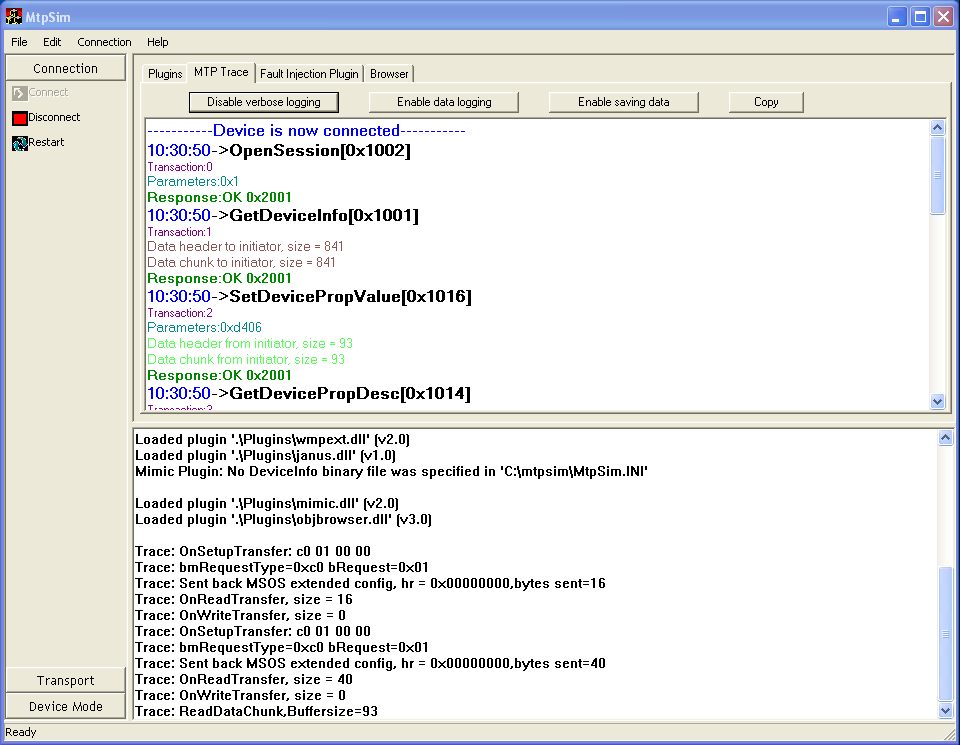


Table 1. MtpSim User Controls \_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| MtpSIM Configuration | Location | Purpose |
| Connection | Left control tabs and menu options. | Connect, disconnect, or restart MtpSim by using the currently selected configuration. You can change the transport type on the **Transport** tab and change the test mode on the **Device Mode** tab. |
| Transport | Left control tabs and menu options. | Select the transport to be used. |
| Device mode | Left control tabs and menu options. | Normal Mode should be used. Test Mode is used for internal Microsoft testing. |
| Output windows | Lower-right output window. | This window displays redirected output from stdout. Any output from a printf statement such as trace—from either MTPSIm or a plug-in—appears in this window. The output is also saved as mtpsimui.log automatically under the c:\mtpsim directory. |
| Plug-ins | Upper-right plug-in property sheet window. | Plug-in UI is implemented as a dialog box and is displayed by MtpSimUI as a property page. Use tabs to navigate through different plug-ins. To configure which plug-in to load, modify mtpsim.ini and add the new plug-in entry into the [Plugins] section. Future versions are expected to provide support for plug-in load/unload menu options |

## Troubleshooting

If you experience any problems during installation, see Appendix B for troubleshooting the problem.

# Device and Service Capability Configuration

You can control the capabilities that the MtpSim simulator reports to an initiator by using an XML file instead of a hard-coded approach (see Appendix F). Device capabilities are controlled by DeviceSummary.xml, and service capabilities are controlled by Services.xml.

The DeviceSummary.xml file enables you to specify the DeviceInfo, StorageInfo, DevicePropDesc, and ObjectPropDesc datasets.

The Services.xml file enables you to specify configurations that are related to device services, including service-supported formats, methods, and events.

You are not required to create the XML configuration files manually. Instead, you can use the MtpDeviceInspector tool to generate the XML file for use in the simulator from any connected WPD device.

Note  Currently, only device configuration XML can be generated.

To load a particular device configuration XML file, on the **Edit** menu, click **LoadDeviceConfig** and then choose the profile.

# Device Stage Configuration

MtpSim lets you change the Device Stage profile during testing. This feature gives you the flexibility to create different scenarios for device testing.

## Default Device Stage Configuration

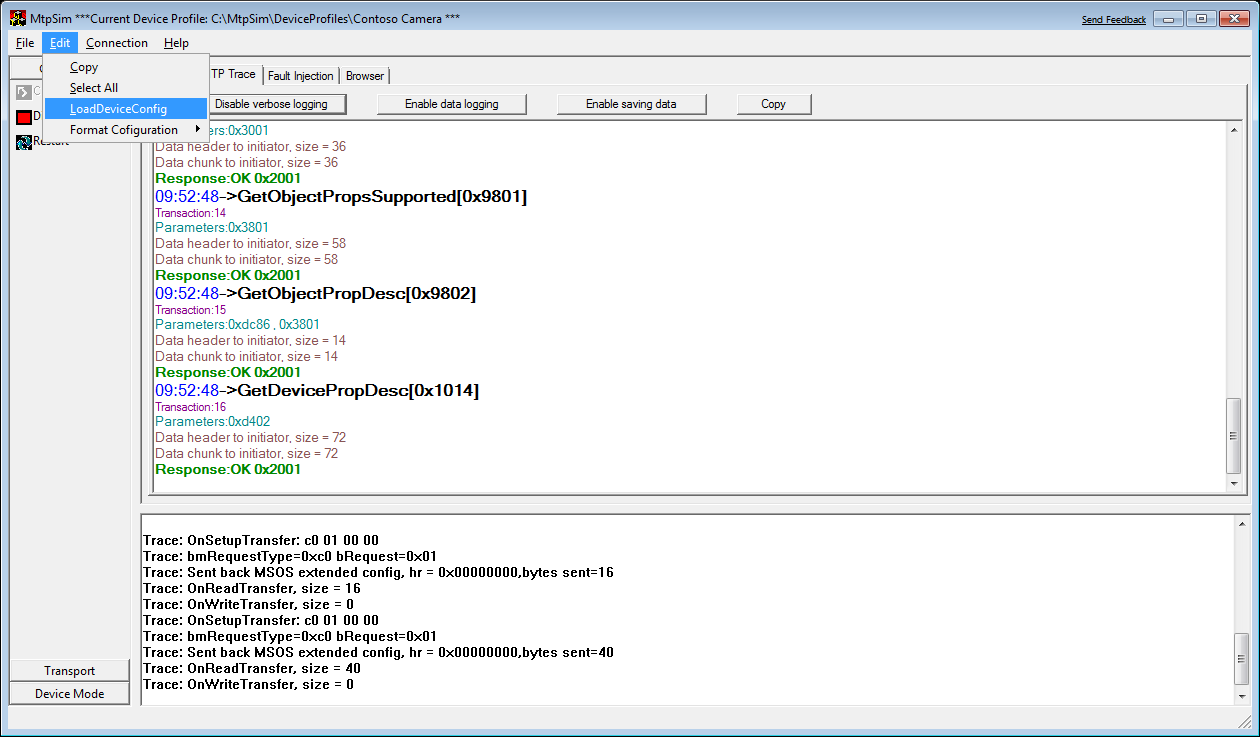
MtpSim starts by loading a default Device Stage Configuration metadata package.

## Enhanced Device Stage Configuration

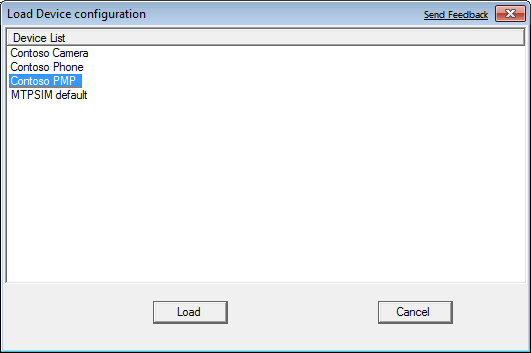
MtpSim can emulate a digital camera, a mobile phone, or a portable media player device experience, with three additional device metadata packages for an enhanced device and demonstration/scenario verification experience. The default behavior of the simulator does not change.

To enable the Enhanced Device Stage Configuration

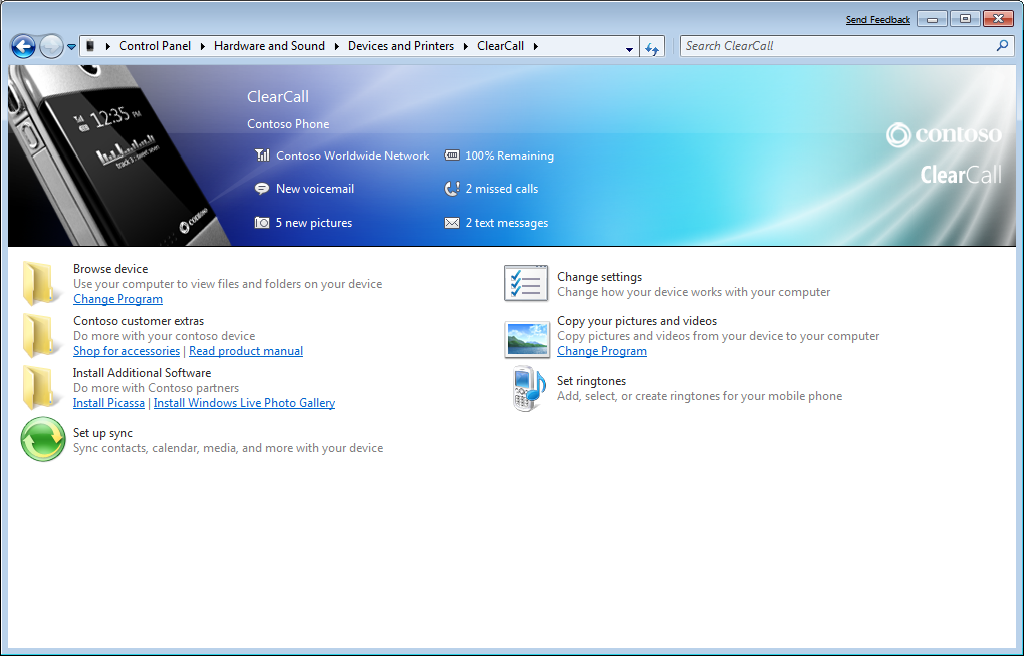
1. On the **Edit** menu, click **LoadDeviceConfig**.



2. Click your configuration type: **Camera**, **Phone**, or **PMP**.



3. Click **Load**, and then click **Connect**.



## Replacing Device Stage with Third-Party Profiles

You can replace the device metadata packages in the simulator with your custom packages by copying the hardware IDs from the MtpSim simulator and replacing the IDs in your metadata packages with them.

MTP Device Simulator supports hardware emulation by loading device configurations that are specified in XML documents. You can test a new device experience by using the simulator with the appropriate device profile. You can use the generic profiles that are included with the simulator with minor modifications as described in the following steps.

To use the generic profiles in MtpSim

1. Open the Device Profiles directory in the MtpSim folder (for example, %systemdrive\mtpsim\deviceprofiles).

2. Copy one of the profiles of interest (for example, a generic camera, telephone, or protected media path—PMP—profile) that will be used as the emulated-device profile.

3. Modify the DeviceSummary.xml file as follows:

The DeviceSummary.xml document contains the properties, events, formats, and methods that the device supports. Several nodes must be adjusted: HardwareID, ModelID (DeviceProperty code-"0xD302"), and device information such as model name, serial number, and so on.

HardwareID

The HardwareID is a legacy property that is used to identify the Vendor, Product, and Revision of the device. Change the numbers in this node to reflect the Hardware ID of the device that you are trying to emulate, as shown in the following example:

<HardwareID VendorID="0x045E" ProductID="0x1113" RevisionID="0x0100" />

DeviceProperty

DeviceProperty is the so-called "Model ID," which is a new property in Windows 7. ModelID is a globally unique identifier (GUID) that uniquely identifies a type or model of device. It is not as specific as a serial number, but should be used to identify devices that share the same device experience (for example all pink-colored telephones or 120-GB branded PMPs). A ModelID is required in the XML *only* if the device experience metadata specifies ModelID.

Note  You must remove ModelID if it is not used in the metadata because the Device Stage Service uses ModelID to match the metadata and a mismatch in either the DeviceSummary or device metadata prevents the experience from loading.

The DeviceProperty section of DeviceSummary.xml resembles the following:

<DeviceProperty code="0xD302" name="VendorExtension" datatype="0x000A" access="0x00">

<DefaultValue>{00000000-0000-0000-0000-000000000000}</DefaultValue>

<CurrentValue>{a8227263-16cc-4bff-855e-a63a3ef746d2}</CurrentValue>

<Form flag="0x00" name="None"/>

</DeviceProperty>

The following nodes are at the bottom of the XML file and are all next to each another. They contain the manufacturer, model name, device version, and serial number information that are advertised to the shell namespace and the device manager. Because all these values are strings, you can include any format that you want in these nodes.

* <Manufacturer value="Contoso"/>
* <Model value="SureShot"/>
* <DeviceVersion value="V3.135.B85"/>
* <SerialNumber value="201984-MNP-24"/>

4. Modify the Services.xml file as follows:

The Services.xml document contains the properties, events, formats, and methods for each service that the device supports. One node must be adjusted in the Metadata Service.

The CabFileInformation node is the property of the metadata service that specifies the metadata package file. This should be copied from the %systemdrive%\mtpsim\cabfiles directory, to the metadata service store (for example, C:\mtpsim\Contoso\_ClearCall\_V1.10.22.2008.04\_MNP08-201984‑24\  
ServiceStore-0x100004). The name of the metadata package file is a unique file name GUID that ends in the .devicemetadata-ms file name extension. The Locale property specifies to which language locale the metadata package file should be unpacked. The Default property specifies whether the default metadata file is used for this device experience (1 == true, 0 == false). The Guid property is the ContentID:

<!--MetaDataInformation Node-->

<MetaDataInformation>

<CabFileInfo Name="ffa2a2a2-0000-0000-0000-ff0000000000.devicemetadata-ms" Locale="en-us" Default="1" Guid="{ffa2a2a2-0000-0000-0000-ff0000000000}"/>

</MetaDataInformation>

# Plug-in Mechanism

Plug-ins are extensible DLLs that are built by using the MtpSim plug-in Software Development Kit (SDK). Plug-ins enable users to extend the capabilities of the simulator and add support for newer commands that the tool can handle. The INI configuration file can be used to define the plug-ins that MtpSim must load. The INI configuration is controlled by using the MtpSim.ini file, which is read at startup to configure certain parts of the simulator. For details, see Appendix E.

## Loading Plug-ins

Plug-ins are loaded when an initiator connects to the simulator. Plug-ins that should be loaded are listed in the INI configuration file, MtpSim.ini. Each plug-in is loaded dynamically and, if a plug-in fails to load for any reason, it is skipped and does not cause the simulator to fail. Simply ensure that the simulator can access the path of the plug-in . We recommend that you do not modify the default plug-in settings.

## Unloading Plug-ins

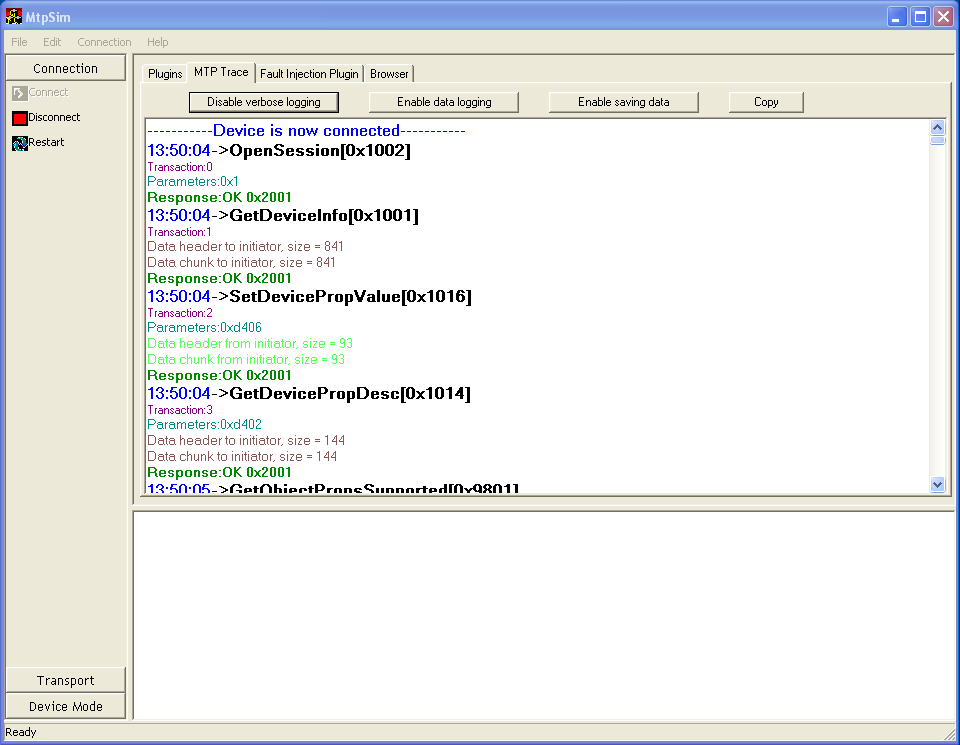
Plug-ins are unloaded when an initiator disconnects from the simulator.

# MtpSim Built-in Plug-ins

MtpSim ships with several plug-ins that improve the usability during simulation. These plug-ins enable tracing and interacting the simulator. Each plug-in is detailed in the following sections.

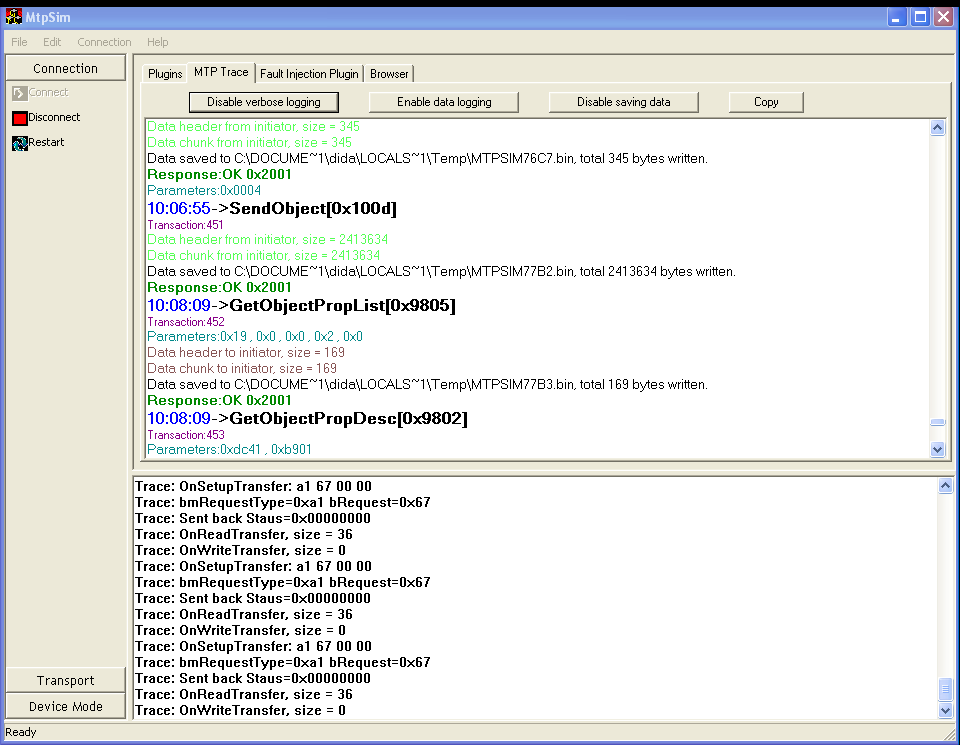
## MTP Trace Plug-in (optrace.dll)

You can use the MTP Trace plug-in to log the MTP commands between an initiator and MtpSim. You can also use it to display binary data and save it to a temporary file. The output can also be logged into a text file that is named MTPTrace.log.



The following describes the controls for the MTP Trace plug-in:

* **Disable verbose logging**Use this button to disable tracing; the button’s text switches to **Enable verbose logging** and the button can be used to enable tracing again.
* **Enable data logging**Use this button to enable logging the data in binary format; the button’s text changes to **Disable data logging** and the button can be used to disable binary data tracing.
* **Enable saving data**Use this button to enable saving data into a temporary file; the button’s text changes to **Disable saving data** and it can be used to disable temporary file saving. The data file is saved to your temp directory with the file name prefix MtpSim.
* **Copy**Use this button to copy all or selected screen output to the clipboard.

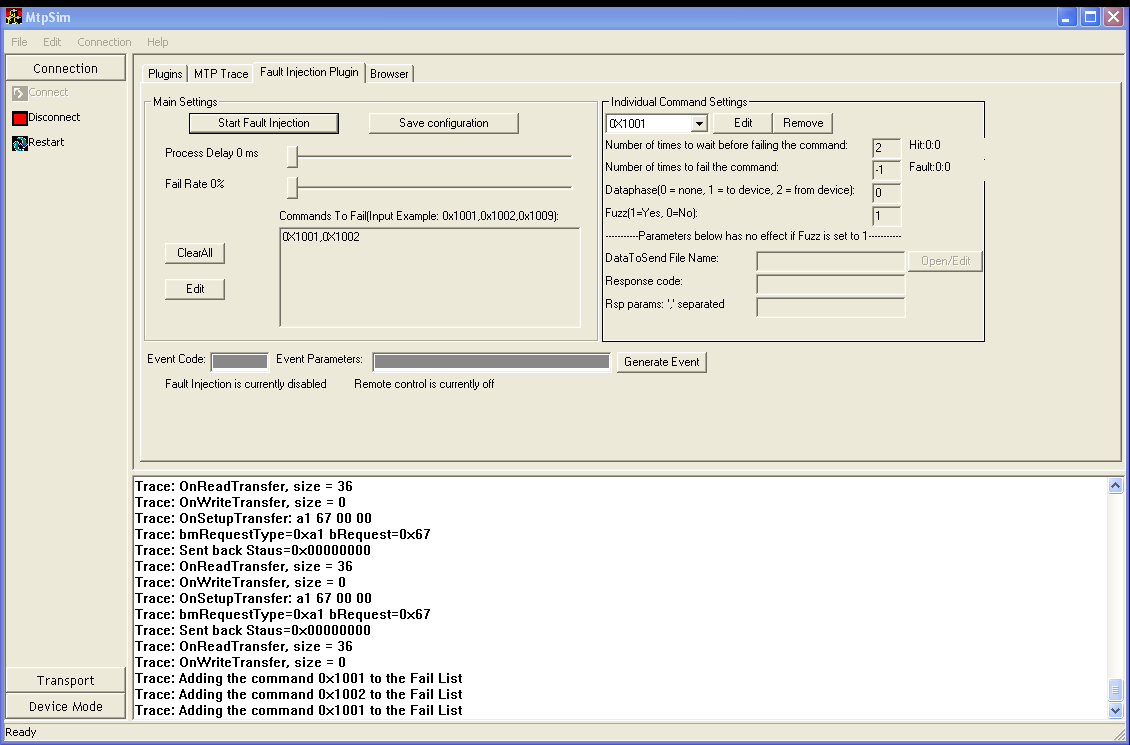


Note  If the Trace plug-in is loaded from console program mtpsim.exe, its UI does not work but the regular trace data is output to the console.

Note  You can decrease MtpSim performance while Verbose Logging is enabled. If you are running tests for performance, you should disable Verbose Logging.

## Fault Injection Plug-in (injfault.dll)

You can use the Fault Injection plug-in to randomly or programmatically insert failures or fuzzed response/data into the initiator. The user can set when and how to fail a particular command. The Fault Injection plug-in saves its configuration in the injFault.ini file. Initially, when the plug-in is loaded, it reads the injFault.ini and populates its configuration UI. This plug-in is not enabled by default. It is primarily intended to test MTP initiators.

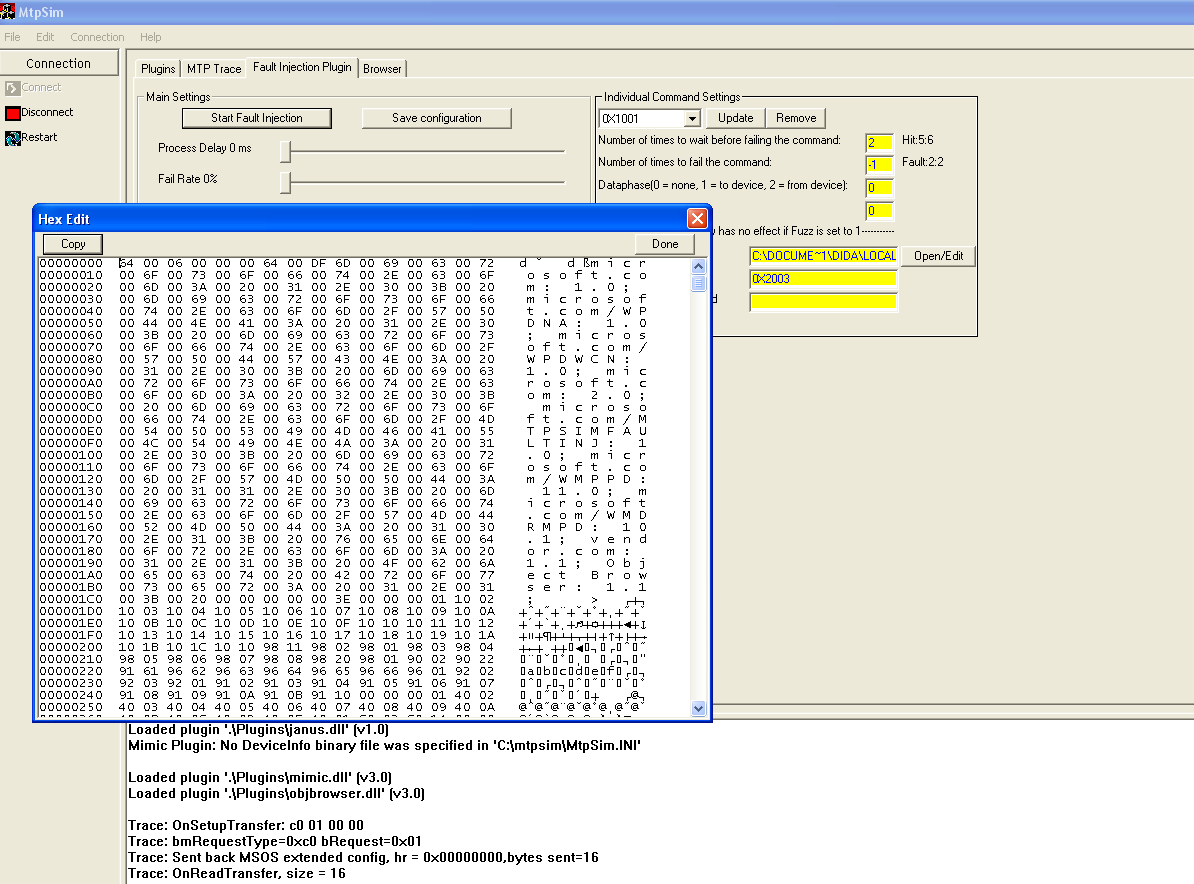


The following describes the Fault Injection plug-in UI usage:

* **Start Fault Injection**  
  Fault injection takes effect after this button is clicked, and its text changes to **Stop Fault Injection**.
* **Save configuration**Save configuration saves the current fault injection configuration to the injfault.ini file. The saved configuration is loaded automatically when the plug-in is started next time.
* **Process Delay**This slide bar can be used to introduce a processing delay between two command processing operations, thus slowing down the MtpSim. Eventually, the simulator supports the introduction of delay in data transfer in or out of the simulator.
* **Fail Rate**This slide bar sets the random failure rate for MTP command processing operations. For example, if the Fail Rate is set to 10%, 1 out of 10 commands fails. The commands to fail and how they fail are selected randomly.
* **Commands To Fail**You can also set specific commands to fail by adding them to this text box. Click **Edit**, and change the text in the **Commands To Fail** list so that it includes the command that you want to fail. Click **Update** to let the plug‑in validate and accept the command-list input. The command list consists of MTP commands and is comma-separated (for example, "0x1001, 0x1002, 0x1009"). The **ClearAll** button can be used to clear the command list. If the command list input is in an invalid format, you are prompted to change it.
* **Individual Command Settings**If a new command is found in the command list, it also appears in the drop-down list in the **Individual Command Settings** pane at the right side. It will have all configurable parameters set to default.

The following explains the parameters:

* **Number of times to wait before failing the command**This command does not fail until the hit count is higher than this number. The default value is 0.
* **Number of times to fail the command**This command fails for a specified number of times. The default value is -1, which means always fail.
* **DataPhase**Set this to specify whether the command processing must follow any data. It can to set to None (0), to device (1), or from device (2). The plug-in must know this value to stay synchronized with the initiator. The default value for this parameter is 0.
* **Fuzz**When set to Yes (1), the data sent back to the initiator and the response code is random. If you set it to No (0), the following three parameters can be edited:
* **DataToSendFileName**The user can specify the full path and file name of a binary data file to send back to the initiator as the response data. Click **Open/Edit** to open a simple binary editing dialog box to let you manually modify the data, as shown in the following screen shot. Note that any modifications that you suggest are applied immediately to the binary file. Therefore, back up your file before you make any changes.

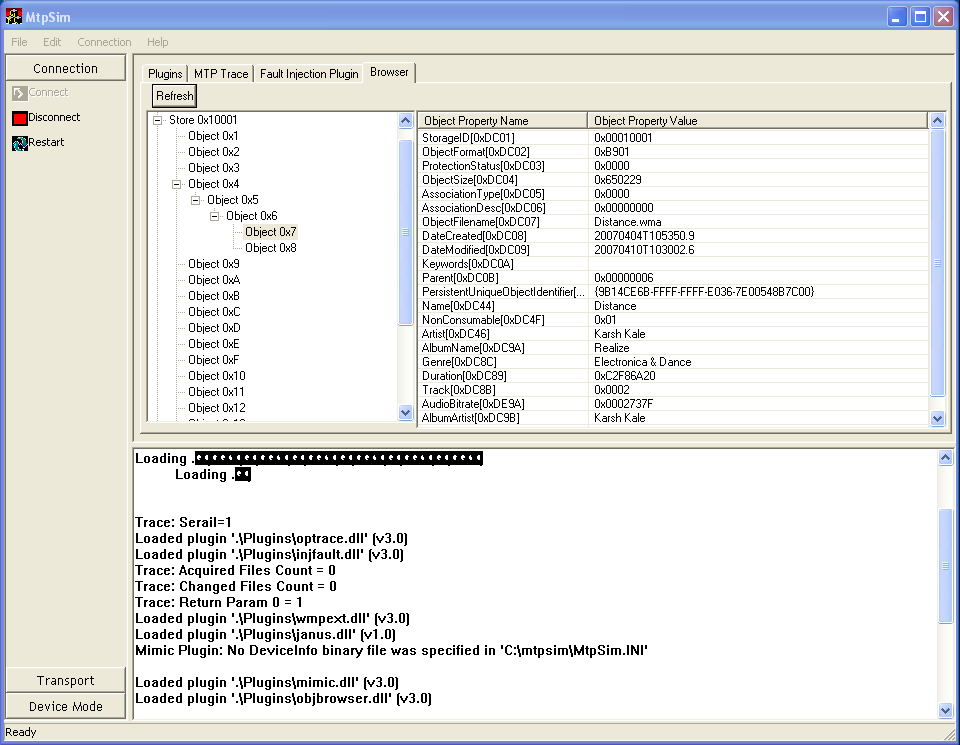


* **Response code**You can specify a failure response code in hexadecimal in this parameter.
* **Rsp Params**The plug-in returns the input comma-separated response parameters to the initiator.

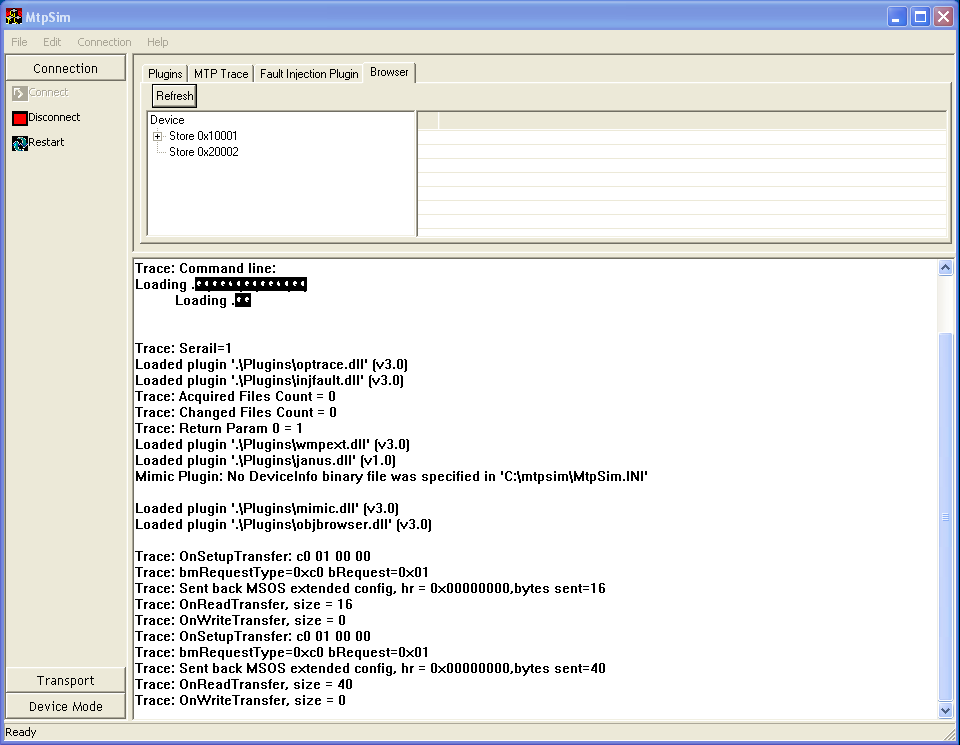
You can also manually generate events to be sent to the initiator by using the **Event Code** and **Event Parameters** fields and the **Generate Event** button. Note that you should separate parameters by commas. In addition to configuring and enabling fault injection from UI, you can also program the simulator to receive commands from the initiator (such as wpdcmd or test running on initiator) by using the custom MTP commands. The custom MTP commands and their usage are defined in Appendix G.

## Browser Plug-in (objbrowser.dll)

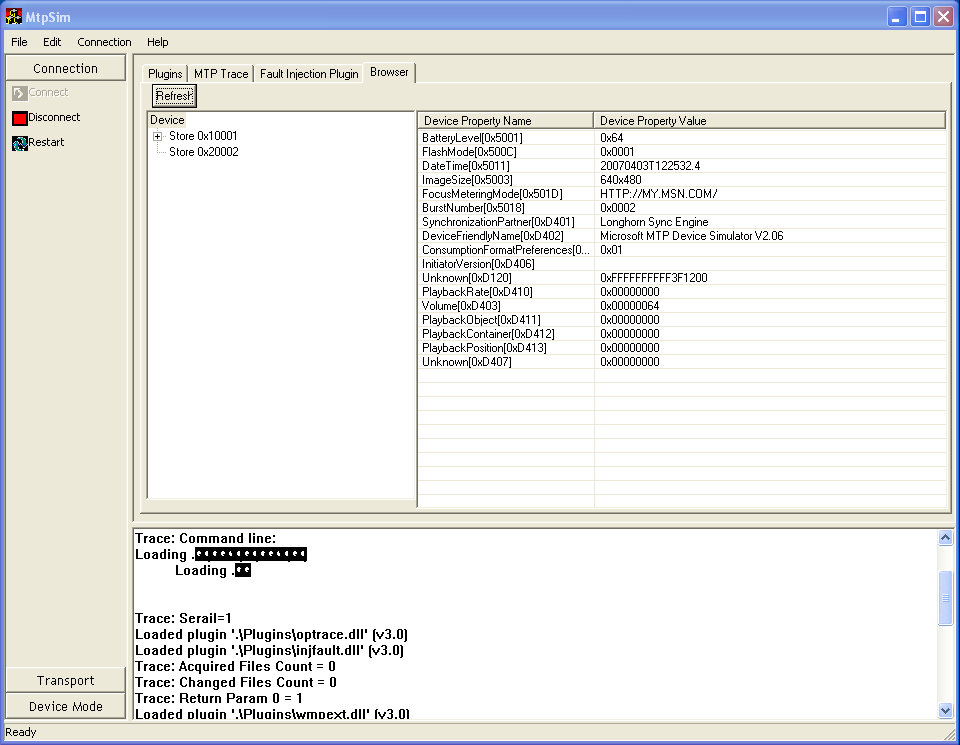
This plug-in exposes the MtpSim internal object store to the user. It exposes device properties, storage information, the object hierarchy, and object properties through a list table. This plug-in is enabled by default.



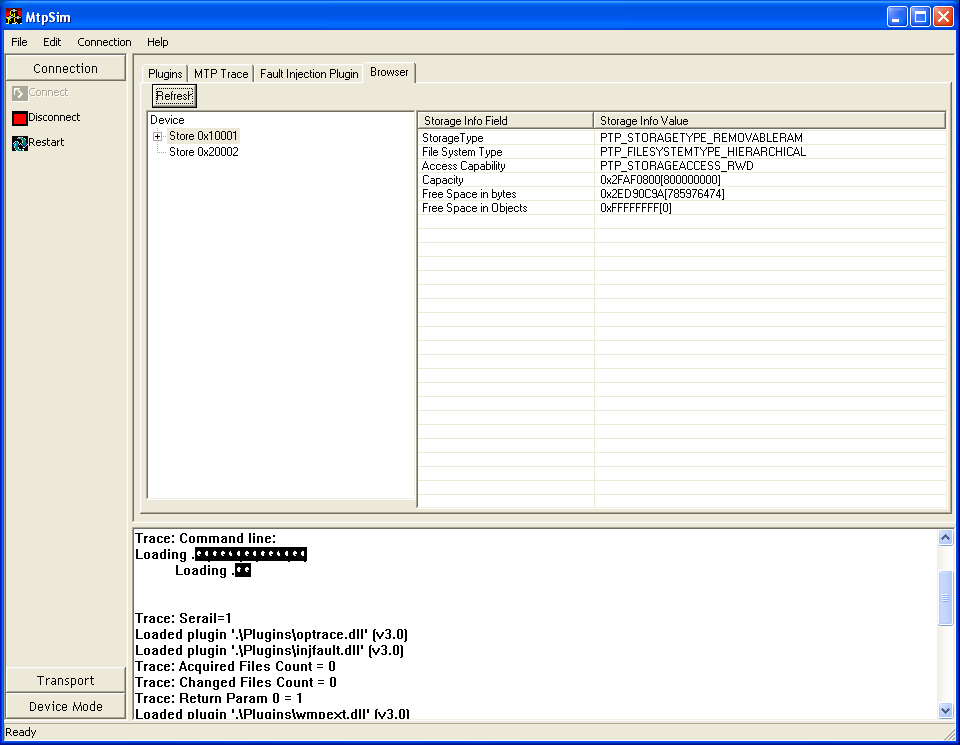
After MtpSim is connected to the initiator, the browser plug-in is initialized and shows the device and its storage.



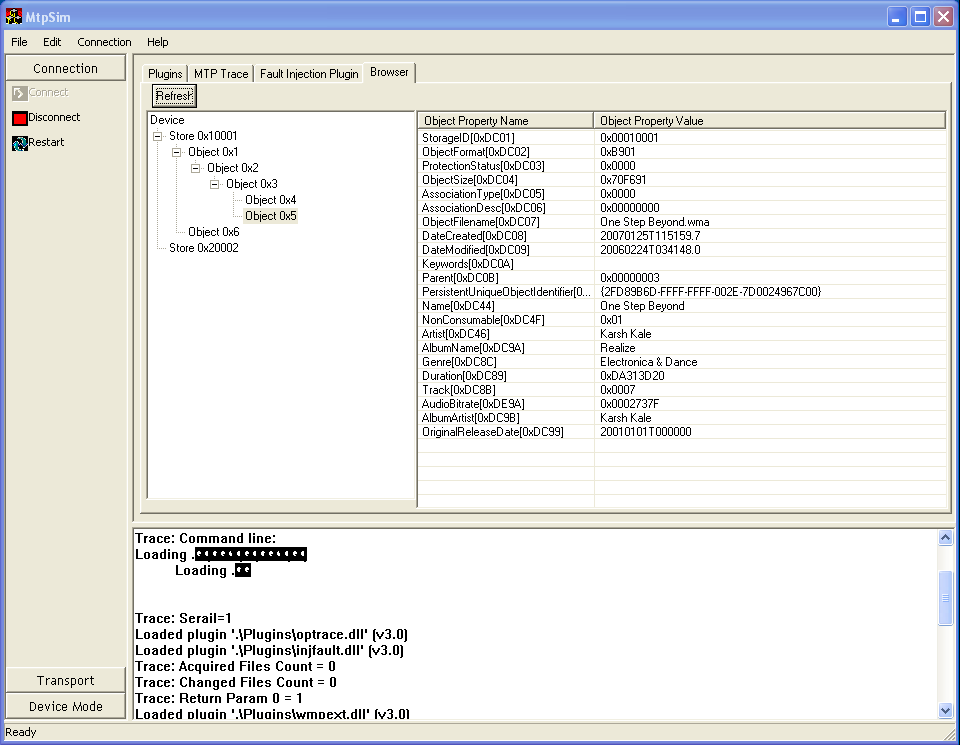
Click **Device** to show all the supported device properties and their values.



Click the name of a storage to show the storage information fields and their values.



You can expand the tree items to see the object hierarchy, and you can click objects to display their supported object properties and values.



Right-click a device, storage, or object to see a **Delete** option, or use the Delete key to delete that item. Therefore, an event is sent to the initiator about the deletion. Other options are **Edit Property**, **Add Property**, and **Modify Property**. A corresponding MTP event (such as **Object Changed**) is generated and sent to the host. The modified properties are saved when the simulator is closed.

# Resources

* To learn more about MTP, download the MTP Specification at   
  <http://www.usb.org/developers/devclass_docs/MTP_1.0.zip>.
* For more information on developing MTP Devices for Windows 7, visit   
  <http://www.microsoft.com/whdc/device/wpd/default.mspx>.
* For more information on device services, look for the following two papers in the WinHEC section of the WHDC Web site at <http://www.microsoft.com/whdc/winhec/2008/papers.mspx>.
* MTP Device Services Extensions
* WPD Device Service Definitions
* For help with the MTP Simulator, send e-mail to [askmtp@microsoft.com](mailto:askmtp@microsoft.com).

# Appendix A. Windows 7 MTP Device Services Extensions

MtpSim supports all the new device services in Windows 7. This appendix describes MtpSim’s specific functionality for each service.

|  |  |
| --- | --- |
| Service | Description |
| Contacts Service | Contains data for contacts, and uses a Synchronization Service to synchronize with the PC. A device can have multiple contact stores, depending on how contacts are organized on the device. Examples of contact stores include subscriber identity module (SIM) card contacts, work contacts, personal contacts, and so on. |
| Calendar Service | Contains data for the calendar, and uses a Synchronization Service to synchronize with the PC. |
| Device Status Service | Reports various status properties of the device to the PC. When the value for a status property is changed, the value is sent to the PC. Note that device and PC performance are affected by your choice of update frequencies. All properties are queried upon device connect for initial state. When a property changes its value, the event is noted and the PC will requery that property. |
| Hints Service | Lets the device select preferred storage locations for various content types in legacy storages. |
| Windows Device Stage Metadata Service | Allows the delivery of Device Stage metadata by the device on first-connect. This is a faster alternative to the download of Device Stage metadata from the Windows Media Information Service (WMIS) data service and the only way to obtain device metadata when no Internet connection at driver installation time exists. This device metadata is delivered to the Metadata Retrieval Client (MRC) Store before Display Object processing. |
| Ringtones Service | Stores ringtones without any folder hierarchy on the target storage. Capabilities declared for the ringtone file formats may differ from capabilities for the media storages (bit rate, duration, and so on), and they should be declared in the service. The Ringtones Service supports MP3, WMA, and AAC formats. |
| Enumeration Synchronization Service | An abstract service that other services can use. Enumeration-based synchronization is fully driven by Windows and requires no special support for an MTP responder. Synchronization occurs in three stages: enumeration of objects, comparison of changes, and updates. |
| Anchor Sync Service | An abstract service that is defined to enable other services to opt in to a synchronized relationship by using Anchor Sync semantics. As an abstract service, the actual implementation of the properties, formats, and methods in the service are the responsibility of the "concrete" service that uses it. |

# Appendix B. Troubleshooting

## Test Certificate Requirement

For the MtpSim to run in "soft" mode, the device must be installed by using a custom test-signed INF and custom drivers. Because the test certificate is not present on RTM builds, the MtpSim driver package remains in not-trusted status even though it was test-signed. This blocks driver installation on Windows Vista x86 and x64 platforms.

The workaround requires installing the test certificate on the destination computer. The installation script tries to perform this action during installation. The steps to perform this manually are:

1. Look for the test certificate file—testroot.cer—provided in the Device Enabling Kit.

2. Run the certutil tool to install the test certificate by using the following command:

certutil -addstore Root testroot.cer

This installs the test certificate, which lets you install the MtpSim drivers. This step may require a reboot to complete the installation.

## Restarting MtpSim after Ending the Process

If the simulator is shut down forcibly (by using Task Manager) or by closing an attached debugger, it cannot issue an unplug notification in "soft" or IP mode.

To enable the simulator to start in "soft" mode when it was previously killed in "soft" mode, the previous instance must be unplugged from the bus enumeration. You can do this by running the enum.exe tool (in the c:\mtpsim\pipedrivers folder) with the **Enum.exe u 1** command.

If the MtpSim failed to connect in SoftUSB mode, do the following:

1. Check whether Soft EHCI is present in device manager under USB controllers. If not, run:

%programfiles%\dsf\dsf\softehci\softehcicfg.exe/install.

2. Run **cleardevice.cmd** under the mtpsim installation root to clear any existing connections.

3. If the problem still exists, run %programfiles%\dsf\ dsf\softehci\softehcicfg.exe /remove and then restart the PC and repeat step 1.

To enable the simulator to start in IP mode when it was previously killed in IP mode, previous UPnP references must be removed. To do this, run the following commands:

1. Sc stop upnphost

2. Reg delete "HKLM\Software\Microsoft\UPnP Device Host\Description" /f

3. Reg delete "HKLM\Software\Microsoft\UPnP Device Host\Devices" /f

4. Sc start upnphost

# Appendix C. Command-Line MtpSim Commands

|  |  |
| --- | --- |
| Command | Description |
| /u | Starts MtpSim in USB mode. This requires the Netchip USB card to be installed with the correct drivers. |
| /i | Starts MtpSim in IP mode. This requires IPv6 to be installed on the computer. IPv6 can be installed on Windows XP by using the **ipv6 install** command. |
| /p | Starts MtpSim in "soft"/pipe mode. This is the default value (that is, if /u or /t are not specified, but /p is assumed). This requires the PipeMtpEnum bus enumerator to be installed first. |
| /t | Enables a test interface for the simulator. This enables the simulator to be controlled through a named-pipe channel that is not related to the selected transport layers. This enables the tests to change the state of objects and/or properties on the simulator without going through the initiator. |
| /n | Starts the simulator without turning on the test interface (equivalent to not specifying the /t parameter). This is the default value. |
| /s:SerialNumber | Allows a serial number to be specified. If this parameter is not specified, 1 is used as the serial number. This parameter is useful while starting multiple instances of the simulator in "soft" mode. Distinct serial numbers appear as distinct device instances. |
| /q[:SerialNumber] | Stops a running instance of the simulator. This does not cause a new instance to be started. If multiple instances of the simulator are running, a specific instance can be targeted by using SerialNumber. If a serial number is not specified, 1 is assumed to be the default. If the /q parameter is used, any additional parameters are ignored. For example, mtpsim /q:2 stops the instance of MtpSim started with the serial number 2. This does not apply to MtpSimUI.exe. |
| /c | Forces MtpSim to run in a CPU-intensive mode. If not specified, MtpSim defaults to issuing a Sleep(1) call between checking for commands, to yield the CPU to other processes. If /c is specified, MtpSim does not issue the Sleep(1) call, which causes it to run at 100-percent CPU usage. This can be useful for faster performance and where multiple processors are available on the machine. It does not apply to MtpSimUI.exe. |
| /m | If specified, causes MtpSim to not try to perform a device connect (that is, plugging into the PipeMtpEnum bus or the NetChip card or issuing a UPnP arrival notification). This is useful only for specific tests that can connect directly to the simulator through named pipes or IP without requiring a dev-node. This does not apply to MtpSimUI.exe. |

# Appendix D. Run-Time MtpSim Commands

|  |  |
| --- | --- |
| Command | Description |
| R | Restarts the simulator. This causes the simulator to unplug from the current bus (or issue a UPnP removal notification in IP mode) and reconnect. |
| Q | Stops the simulator. This causes the simulator to unplug from the current bus. |
| L | Lists store contents. This dumps a list of objects that the simulator reports back to the initiator. |
| T | Enables the test interface. This enables the simulator to be controlled through a named-pipe channel that is not related to the current transport layer. |
| N | Enables Normal mode. This turns off the test interface. In ordinary mode, MtpSim cannot receive test commands from clients. |
| U | Switches to the USB transport, if not already connected through USB. MtpSim first unplugs from the current bus and then tries to connect to the USB transport. |
| P | Switches to the "soft"/pipe transport, if not already connected. MtpSim first unplugs from the current bus and then tries to connect to the pipe transport. |
| I | Switches to the IP transport, if not already connected. MtpSim first unplugs from the current bus and then tries to connect to the IP transport. |
| C | Toggles CPU-intensive mode. For more details, see the description of the /c command-line command in Appendix C. |

# Appendix E. INI File Configuration

### **[Device]**

Root Path=<device\_root\_path>

This sets the path in which the simulator should look for the stores and other data.

XmlConfiguration=<path\_to\_xml\_file>

This points to an XML configuration file to load. The XML configuration file is explained in Appendix F. If the configuration file is not specified or if the key is commented out, MtpSim defaults to using a built-in set of device properties, formats, object properties, and so on.

### **[StoreN]**

N is a zero-based index. This section provides information about the storage that MtpSim reports.

Root Path=< store\_root\_path>

This sets the path in which the simulator should look to build an object list for the store. All objects under the specified root path are reported as present on the store when an initiator requests information.

Capacity=<value\_in\_bytes>

This sets the maximum capacity of the store. This is reported in the MaxCapacity field of the StorageInfo dataset for this store. This value can be overridden by the XML configuration file.

StorageID=<Hex value>

This sets the storage ID that is reported by the MtpSim in the GetStorageIDs command for this particular storage. This value can be overridden by the XML configuration file.

### **[ExtensionFormatcodeMapping]**

This section notifies the simulator of the MTP format code that should be reported for an object, based on the file name extension. The simulator already contains a static list that maps well-known extensions to MTP format codes. This section is useful to augment that list by mapping unknown extensions or to override the static list by mapping a well-known extension to a new format code. For example, PLS=0xBA05 indicates that all files that have the file name extension .PLS should be reported as format code 0xBA05.

### **[USBTransport]**

ReportPTPClassDescriptor=[0|1]

This sets whether the USB transport layer (mtpsimus.dll) reports the PTP class descriptor as part of the USB configuration information while connecting to a PC. The PTP class descriptor is primarily used to identify camera devices. If not specified, the default value of this key is assumed to be 0 and the reported class descriptor is a NULL set (Class=0&SubClass=0&Protocol=0).

ReportMSOSDescriptor=[0|1]

This sets whether the USB transport layer (mtpsimus.dll) reports the Microsoft OS extended descriptor as part of the USB configuration information while connecting to a PC. PCs that have Windows Media Player 10 or later probe for the MSOS extended descriptor. If the descriptor is present and set to a specific descriptor, the PC treats the device as a media-player device. If the descriptor is not specified, the default value is assumed to be 1 and the MSOS extended descriptor is reported as the MTP value. You can find specific MTP OS Descriptor values in “Portable Device Installation Considerations” that is part of the Windows Portable Device Enabling Kit.

Note  To configure MtpSim to install as a pure still-image device, set ReportPTPClassDescriptor=1 and ReportMSOSDescriptor=0.

### **[NetworkAssociation]**

This section lists the EUI64 IDs of PCs that are associated with the simulator for UPnP/IP connection. The format of the key is a string representation of the EUI64 GUID with the value controlling whether the PC was associated. For example, {00000000-0000-0000-FFFF-000D561F7364}=1 indicates that the specified encoded MAC address was associated with the PC.

AuthenticateInitiator=[0|1]

This specifies whether the simulator must verify that the initiator that is connected over the PTP/IP protocol was previously associated with it. If not specified, the default value is 0, which means that the simulator is not required to verify the previous association with the initiator.

# Appendix F. XML Configuration File Details

The XML configuration file controls basic settings of the simulator and enables or disables each of its components. The following list describes the parts of the XML configuration file.

<Device> node

The main root node that contains the other nodes.

**<DeviceInfo> node**

The same information that would be present in the DeviceInfo dataset as per the MTP Specification (see section 5.1.1).

**<StandardVersion value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this field.

**<MTPVendorExtensionID value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this field.

**<MTPVersion value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this field.

**<MTPExtensions value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this field.

**<FunctionalMode value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this field.

**<OperationsSupported>**

Corresponds to the OperationsSupported array field in the DeviceInfo dataset. The node consists of child nodes, each of which corresponds to an array member.

**<Operation code="…" name="…"/>**

A stand-alone node whose **code** attribute specifies the MTP operation code that should be listed in the OperationsSupported array. The **name** attribute is present only for information and is not processed by the simulator.

**<EventsSupported>**

Corresponds to the EventsSupported array field in the DeviceInfo dataset. The node consists of child nodes, each of which corresponds to an array member.

**<Event code="…" name="…"/>**

A stand-alone node whose **code** attribute specifies the MTP event code that should be listed in the EventsSupported array. The **name** attribute is present only for information and is not processed by the simulator.

**<DevicePropertiesSupported>**

Corresponds to the DevicePropertiesSupported array field in the DeviceInfo dataset. The node consists of child nodes, each of which corresponds to an array member.

**<DeviceProperty code="…" name="…" datatype="…" access="…">**

The **code** attribute specifies the MTP device property code that should be listed in the DevicePropertiesSupported array. The **name** attribute is present only for information and is not processed by the simulator. The node contains additional attributes and nodes that help describe the DevicePropDesc dataset for this property per the MTP Specification (see section 5.1.2.1). The **datatype** attribute specifies the MTP data type that should be reported in the DevicePropDesc dataset. The **access** attribute corresponds to the Get/Set field in the DevicePropDesc dataset.

**<DefaultValue>**

A stand-alone node whose literal value is used as the default value in the DevicePropDesc dataset. If the data type is of an array form, the value of the node can be set to comma-separated number values. If the data type is U/INT128, a string representation of a GUID should be used.  
Example 1.  
**<DeviceProperty code="…" datatype="0x4001" …>  
 <DefaultValue>0x1, 0x2, 0x3, 0x4</DefaultValue>**Example 2.  
**<DeviceProperty code="…" datatype="0x000A" …>  
 <DefaultValue>{00000000-0000-0000-0000-000000000000}</DefaultValue>**

**<CurrentValue>**

A stand-alone node whose literal value is used as the current value in the DevicePropDesc dataset. If the data type is of an array form, the value of the node can be set to comma-separated number values. If the data type is U/INT128, then a string representation of a GUID should be used.   
Example 1.  
**<DeviceProperty code="…" datatype="0x4001" …>  
 <CurrentValue>0x1, 0x2, 0x3, 0x4</ CurrentValue >**Example 2.  
**<DeviceProperty code="…" datatype="0x000A" …>  
 < CurrentValue >{00000000-0000-0000-0000-000000000000}</ CurrentValue >**

**<Form flag="…" name="…">**

The **flag** attribute for the Form node that specifies the form flag that should be listed in the DevicePropDesc dataset. Depending on the value of the **flag** attribute, additional child nodes may exist.

**<Form flag="0x00">**

Form flag None; requires no additional child nodes.

**<Form flag="0x01">**

Corresponds to the Range form that requires all the following qualifying child nodes:  
**<MinimumValue value="…"/>  
<MaximumValue value="…"/>  
<StepSize value="…"/>**

**<Form flag="0x02">**

Corresponds to the Enumeration form that requires zero or more of the following child nodes:  
**<EnumValue value="…"/>**

**<CaptureFormats>**

Corresponds to the CaptureFormats array field in the DeviceInfo dataset. The node consists of child nodes, each of which corresponds to an array member.

**<CaptureFormat code="…" name="…"/>**

A stand-alone node whose **code** attribute specifies the MTP capture format code that should be listed in the CaptureFormats array. The **name** attribute is present only for information and is not processed by the simulator.

**<ObjectFormats>**

Corresponds to the ObjectFormats array field in the DeviceInfo dataset. The node consists of child nodes, each of which corresponds to an array member.

**<ObjectFormat code="…" name="…">**

The **code** attribute that specifies the MTP format code that should be listed in the ObjectFormats array. The **name** attribute is present only for information and is not processed by the simulator. The node contains additional child nodes that help describe the ObjectPropsSupported dataset per the MTP Specification (see section 5.3.2.2) and nested child nodes that help describe the ObjectPropDesc dataset for each supported properties (see section 5.3.2.3).

**<ObjectPropertiesSupported>**

The child nodes of this node correspond to the property code array that is returned in response to a GetObjectPropsSupported call,

**<ObjectProperty code="…" name="…" datatype="…" access="…">**

The **code** attribute specifies the MTP object property code that should be listed in the ObjectPropsSupported array. The **name** attribute is present only for information and is not processed by the simulator. The node contains additional attributes and nodes that help describe the ObjectPropDesc dataset for this property per the MTP Specification (see section 5.3.2.3). The **datatype** attribute specifies the MTP data type that should be reported in the ObjectPropDesc dataset. The **access** attribute corresponds to the Get/Set field in the ObjectPropDesc dataset.

**<DefaultValue>**

A stand-alone node whose literal value is used as the default value in the ObjectPropDesc dataset. If the data type is of an array form, the value of the node can be set to comma-separated number values. If the data type is U/INT128, a string representation of a GUID should be used.  
Example 1.  
**<DeviceProperty code="…" datatype="0x4001" …>  
<DefaultValue>0x1, 0x2,0x3,0x4</DefaultValue>**Example 2.  
**<DeviceProperty code="…" datatype="0x000A" …>  
 <DefaultValue>{00000000-0000-0000-0000-000000000000}</DefaultValue>**

**<Group Code value="…">**

A node whose value corresponds to the GroupCode field of the ObjectPropDesc dataset.

**<Form flag="…" name="…">**

The **flag** attribute for the Form node that specifies the form flag that should be listed in the ObjectPropDesc dataset. Depending on the value of the **flag** attribute, additional required child nodes may be required.

**<Form flag="0x00">**

Form flag None; requires no additional child nodes:

**<Form flag="0x01">**

Corresponds to the Range form that requires all the following qualifying child nodes:  
**<MinimumValue value="…"/>  
<MaximumValue value="…"/>  
<StepSize value="…"/>**

**<Form flag="0x02">**

Corresponds to the Enumeration form that requires zero or more of the following child nodes:  
<EnumValue value="…"/>

**<Form flag="0x03">**

Corresponds to the DateTime form that does not require additional child nodes.

**<Form flag="0x04">**

Corresponds to the FixedLengthArray form that can be further qualified with the following optional child node:  
<Length value="…"/>

**<Form flag="0x05">**

Corresponds to the RegularExpression form that can be further qualified with the following optional child node:  
<RegEx value="…"/>

**<Form flag="0x06">**

Corresponds to the ByteArray form that can be further qualified with the following optional child node:  
<MaxLength value="…"/>

**<Form flag="0xFF">**

Corresponds to the LongString form that does not require additional child nodes.

**<InterdependentPropertyDescription>**

An optional node that if present describes the InterdepPropDesc dataset that is returned in response to a GetInterdepPropDesc command for a specific format. See section E.2.3 in the MTP Specification. There can be zero or more Configuration child nodes.

**<Configuration>**

Qualified by its child nodes, which correspond to an array of ObjectPropDesc datasets. There can be one or more ObjectProperty child nodes, each of which corresponds to a specific ObjectPropDesc dataset.

**<ObjectProperty code="…" name="…" datatype="…" access="…">**

The syntax for this node follows the same rules as the <ObjectPropertiesSupported\ObjectProperty> node.

**<Manufacturer value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this DeviceInfo field.

**<Model value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this DeviceInfo field.

**<DeviceVersion value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this DeviceInfo field.

**<SerialNumber value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this DeviceInfo field.

**<Storages>**

Corresponds to the dataset that is returned in response to the GetStorageIds command and contains one or more <StorageInfo> nodes for each supported storage.

**<StorageInfo StorageID="…">**

Corresponds to the StorageInfo dataset per section 5.2.2 in the MTP Specification. The **StorageID** attribute contains the storage ID of the storage and is one of the array elements that are returned for the GetStorageIds call.

**<StorageType value="…" name="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field. The **name** attribute is optional and is not used by the simulator. It is present only to describe the value.

**<FileSystemType value="…" name="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field. The **name** attribute is optional and is not used by the simulator. It is present only to describe the value.

**<AccessCapability value="…" name="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field. The **name** attribute is optional and is not used by the simulator. It is present only to describe the value.

**<MaxCapacity value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field.

**<FreeSpaceInBytes value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field.

**<FreeSpaceInObjects value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field.

**<StorageDescription value="…"/>**

A stand-alone node whose **value** attribute specifies the value that should be used for this StorageInfo field.

**<VolumeIdentifier value="…"/>**

A stand-alone node with the **value** attribute specifying the value that should be used for this StorageInfo field.