MTP Device Services Extension Specification

March 10, 2010

Abstract

The MTP device services extension to the Media Transfer Protocol (MTP) helps an MTP initiator to find and access certain types of content that is stored on a responder (device). These mechanisms provide greater extensibility than the existing datacode mechanisms.

This specification is for device manufacturers that are building MTP-compatible devices. It assumes that the reader is familiar with Windows® Portable Devices (WPD) technology and the *USB Media Transfer Protocol Specification, Revision 1.0*.

This information applies to the Windows 7 operating systems.

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

The current version of this paper is maintained on the Web at:
 <http://www.microsoft.com/whdc/device/wpd/MTPDevServExt_Spec.mspx>

Disclaimer: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

*This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.*

© 2010 Microsoft Corporation. All rights reserved.

Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Change |  |  |  |
| March 10, 2010 | * Expanded description of MTP device services fundamentals.
* Added abstract services.
* Expanded the updated behaviors for non-service MTP operations.
* Updated the extension implementation, including a new GetFormatCapabilities operation and a new UseDeviceStage property.
* Added a “Resources” section.
* Added numbers to the section headings for ease of reference.
 |
| March 17, 2009 | First publication |

Contents

[1.0 Introduction 4](#_Toc255900821)

[2.0 MTP Device Services Fundamentals 4](#_Toc255900822)

[2.1 Numbering Systems 4](#_Toc255900823)

[2.1.1 GUIDs 5](#_Toc255900824)

[2.1.2 PKeys 5](#_Toc255900825)

[2.1.3 Datacodes 6](#_Toc255900826)

[2.2 Names 6](#_Toc255900827)

[2.3 Abstract Services 7](#_Toc255900828)

[2.4 Core MTP 1.0 Requirements 7](#_Toc255900829)

[2.4.1 Operations 7](#_Toc255900830)

[2.4.2 Formats 8](#_Toc255900831)

[2.4.3 Object Properties 8](#_Toc255900832)

[2.4.4 Functionality Not Used in Services 8](#_Toc255900833)

[3.0 Updated Behaviors for Non-Service MTP Operations 8](#_Toc255900834)

[3.1 Unsupported Operations 8](#_Toc255900835)

[3.2 GetObjectPropList 8](#_Toc255900836)

[3.3 DeleteObjectPropList 9](#_Toc255900837)

[3.4 SetObjectPropList/SendObjectPropList 9](#_Toc255900838)

[3.5 UpdateObjectPropList 10](#_Toc255900839)

[3.6 GetFormatCapabilities 10](#_Toc255900840)

[3.7 Maintaining Compatibility with MTP 1.0 10](#_Toc255900841)

[3.8 Maintaining Compatibility with PTP and Other non-Windows
 MTP Initiators 10](#_Toc255900842)

[4.0 MTP Device Services Extension Implementation 11](#_Toc255900843)

[4.1 MTP Extension Identification 11](#_Toc255900844)

[4.2 Extension Operations 11](#_Toc255900845)

[4.2.1 GetServiceIDs 11](#_Toc255900846)

[4.2.2 GetServiceInfo 12](#_Toc255900847)

[4.2.3 GetServiceCapabilities 15](#_Toc255900848)

[4.2.4 GetServicePropDesc 16](#_Toc255900849)

[4.2.5 GetServicePropList 17](#_Toc255900850)

[4.2.6 SetServicePropList 19](#_Toc255900851)

[4.2.7 UpdateObjectPropList 19](#_Toc255900852)

[4.2.8 DeleteObjectPropList 21](#_Toc255900853)

[4.2.9 DeleteServicePropList 22](#_Toc255900854)

[4.2.10 GetFormatCapabilities 24](#_Toc255900855)

[4.3 Extension Responses 25](#_Toc255900856)

[4.4 Extension Device Properties 25](#_Toc255900857)

[4.4.1 Functional ID 25](#_Toc255900858)

[4.4.2 Model ID 26](#_Toc255900859)

[4.4.3 UseDeviceStage Flag 26](#_Toc255900860)

[4.5 Extension Events 27](#_Toc255900861)

[4.5.1 ServiceAdded Event 27](#_Toc255900862)

[4.5.2 ServiceRemoved Event 27](#_Toc255900863)

[4.5.3 ServicePropChanged Event 27](#_Toc255900864)

[4.5.4 MethodComplete Event 27](#_Toc255900865)

[4.6 Extension Object Properties 28](#_Toc255900866)

[4.6.1 Object Property Forms 28](#_Toc255900867)

[5.0 Resources 30](#_Toc255900868)

# 1.0 Introduction

The Media Transfer Protocol (MTP) device services extension helps an MTP initiator find and access certain types of content that is stored on a responder (device). Today’s devices have content that is distributed throughout the device, which makes it difficult for MTP initiators such as the Windows® operating system to determine the location of the useful assets and how to use them. By using an extension to MTP, an MTP initiator (such as a PC) can locate, consume, and interact with useful content and services on the device (an MTP responder). This extension is particularly useful for accessing device content that is not based on file system data, settings, and restricted capabilities. This feature also enables the MTP device architecture to support future platform extensibility.

Device manufacturers can use this specification to:

* Support a set of MTP device services in their device, such as the MTP device services for Windows 7.
* Define a vendor-specific MTP extension.
* Define a vendor-specific MTP service.

This specification outlines the operation codes that the initiator and responder must implement to effectively support the MTP device services extension. Although MTP already supports enumeration by format, the extension mechanisms provide greater flexibility for applications that handle specific content types.

This extension moves much of MTP communication to a system that is based on globally unique identifiers (GUIDs). GUIDs provide greater extensibility than the existing datacode mechanisms, and eliminate the possibility of datacode collision to ensure that third-party extensions to MTP do not conflict with other extensions.

To increase ease of use, this extension adds computer-readable names to service objects. These names can be promoted to script-style languages along with objects that these services represent.

# 2.0 MTP Device Services Fundamentals

The following sections describe important concepts and mechanisms of MTP device services.

## 2.1 Numbering Systems

The MTP device services definition contains three numbering systems: GUIDs, property keys (PKeys), and datacodes. GUIDs and PKeys are defined by device service definitions. Datacodes are generated by the device manufacturer during implementation or at runtime.

The *USB Media Transfer Protocol Specification, Revision 1.0* (the MTP specification) uses datacodes such as Operation Code, Format Code, and Property Code. The MTP device services extension uses GUIDs to expand the limited MTP communication mechanism. MTP device services also use names with each GUID and PKey to provide easy access for programming languages.

### 2.1.1 GUIDs

MTP device services use GUIDs to identify services, events, and formats. MTP device services GUIDs must be:

* 128-bit values that are formed as tightly packed structures of one unsigned 32‑bit value, three unsigned 16-bit values, and six unsigned 8-bit values. The following C‑style definition is appropriate for a GUID:

typedef struct \_PSLGUID

{

 UINT32 dwValue;

 UINT16 wValue1;

 UINT16 wValue2;

 UINT8 rgbData[8]

} PSLGUID;

* Transmitted over MTP by using little-endian byte ordering.
* Expressed as a UInt128 value when they are used as the result of a property.
* Generated by a reliable GUID generator and guaranteed to be unique across all devices and all applications.
* Responsible for the meaning and functionality of a service, property, method, or object format, as published in a device service definition.

### 2.1.2 PKeys

MTP device services use PKeys to identify object properties, service properties, and method parameters. A PKey consists of a GUID namespace and a 32-bit unsigned integer ID. Where possible, device service implementers should use existing, well-defined namespaces to promote interoperability.

MTP device PKeys must conform to the following rules:

* PKey Namespaces:

Are encoded as standard GUIDs.

Must be generated by a reliable GUID generator and must be guaranteed to be unique across all devices and applications.

Are used to partition properties into related groups.

* PKey IDs:

Must be unique within a namespace, but can be reused between namespaces.

Are not required to be random.

Do not use the values 0 and 1; PKey IDs 0 and 1 are reserved.

Are assigned only by the creator or manager of the namespace and must not be added to existing namespaces.

* PKey Namespaces and PKey IDs:

Are defined by the device service definition for a particular service.

Contain a well-defined model for additions and updates. The MTP specification process controls the publicly defined PKey Namespaces and PKey IDs that map existing MTP property and format codes into the device services space.

### 2.1.3 Datacodes

Because MTP operations and parameters require 32-bit datacodes, datacodes are still used throughout MTP device services. The real value of each format, object property, device property, service property, event, and so on is set by the corresponding GUID or PKey in the ServiceInfo and CapabilityList datasets.

MTP device datacodes must conform to the following rules:

* Datacodes are 16-bit unsigned integers.
* Datacodes are used for format codes, object property codes, service property codes, event codes, device property codes, and operation codes.
* Datacodes are assigned by the device manufacturer at implementation time or at runtime.
* Format, method, and event codes must be unique across the entire device (all services and storages).
* Format codes for formats that appear only in services are not returned as supported formats in the DeviceInfo dataset.
* Service property codes must be unique within a service.
* Object property codes must be consistent with all objects of the same format:

Format codes must use one set of properties across a device.

For supporting multiple formats with different sets of properties and capabilities, the same format GUID must be used with different format codes that are defined in each service. For example, if the legacy storage uses the MP3 format (0x3009), a ringtones service with different properties for the MP3 format must use a different format code.

* In services, several datacodes can be used for the same PKey or GUID (for example, two format codes for the same format GUID).
* Datacodes must be persistent within a session, but can change between sessions.
* Datacodes can be used as parameters in legacy operations.
* Property codes 0xDC01 and 0xDC02 are reserved and have their standard MTP 1.0 definition.

## 2.2 Names

Names are strings that accompany each GUID and PKey to provide a computer-readable variable name for programming languages. Names should be defined in the definition of each device service and are used primarily for scripting languages.MTP device services names:

* Are strings (with a 255-character maximum).
* Have these mandatory requirements:

Must contain no spaces.

Must be alphanumeric.

Can also contain \_ (underscore), - (hyphen), and @ (”at” symbol).

Must be unique within a service property list, object format definition, method parameter list, or service event list.

* Can follow these additional recommendations:

Be in the English language.

Contain capitalized first letters in concatenated words (ThisIsAnExample).

Include properties that may follow the form *CompanyNamespaceProperty* (such as ContosoStatusBatteryLife).

## 2.3 Abstract Services

MTP device services introduce the concept of an abstract service. Abstract services exist only to define common behaviors that might be shared by other services on the responder. Unlike “concrete” services, abstract services exist only to provide a template of information to which other services refer. The only operation that is supported on an abstract service is GetServiceInfo.

A concrete service that uses an abstract service must fully support all properties, formats, methods, and events that are defined in the abstract service. This includes operations that query property and format descriptions (GetServicePropDesc and GetServiceCapabilities). However, a concrete service must not contain any properties, formats, methods, or events in its ServiceInfo dataset that are defined in the abstract service.

## 2.4 Core MTP 1.0 Requirements

This section contains information about operations, formats, object properties, and functionality that are used in MTP device services.

### 2.4.1 Operations

To implement MTP device services, a device must implement the following MTP 1.0 operations:

* OpenSession
* CloseSession
* GetStorageIDs
* GetStorageInfo
* GetDeviceInfo
* GetDevicePropDesc

For basic functionality, MTP device services require only the preceding operations and the new operations that are defined in this specification. A device can implement additional operations for legacy storages.

Most device services for Windows 7 also require the following operations:

* GetDevicePropVal
* SetDevicePropVal
* SendObject
* GetObject
* DeleteObject
* GetObjectPropList
* SetObjectPropList
* SendObjectPropList
* UpdateObjectPropList

### 2.4.2 Formats

A device service can include both custom formats and formats from Revision 1.0 of the MTP specification. When devices use MTP 1.0 formats, the devices must use the predefined GUIDs and names that are defined in the header files in the Portable Device Enabling Kit. For more information on the Portable Device Enabling Kit, see “[Resources](#_5.0__Resources)” at the end of this specification.

### 2.4.3 Object Properties

All objects on a device must support the following two object properties:

* StorageID (0xDC01)
* ObjectFormat (0xDC02)

These datacodes (object property codes) must be reserved and supported by all objects, on all storages and services. This requirement is necessary for legacy compatibility and enumeration operations. Additionally, for best performance these properties should be in the same property group. We recommend that these properties be in GroupCode 0x01, but no GroupCode (0x00) is also acceptable.

### 2.4.4 Functionality Not Used in Services

An initiator (such as Windows) that implements MTP device services should not use the operations and properties in the left column of the following table because they have been replaced by the corresponding operations in the right column. Device developers can implement the operations in the left column if doing so reduces the complexity of their MTP implementation. However, an MTP initiator cannot use the old operations in a device service.

|  |  |
| --- | --- |
| Old operations | New operations |
| GetObjectPropDesc | GetServiceCapabilities |
| GetObjectPropsSupported | GetServiceCapabilities |
| GetObjectPropVal | GetObjectPropList |

# 3.0 Updated Behaviors for Non-Service MTP Operations

The MTP device services extension defines MTP operations to improve the overall behavior of MTP-enabled devices in non-service environments. The device can implement these behaviors and operations regardless of whether MTP device services are enabled on the device.

## 3.1 Unsupported Operations

Unlike the traditional “media” behaviors for MTP, introducing device services does not guarantee that all operations always apply to every MTP object or service. When a particular MTP operation does not apply within the current context, responder implementers should use the Operation\_Not\_Supported result code.

## 3.2 GetObjectPropList

To accommodate newer functionality for devices that support MTP device services, we modified the behavior of GetObjectPropList calls to MTP device services to better handle the concept of uninitialized properties. Uninitialized properties are object properties that have no value and no valid default value. Consider the example of a Contact object with a Phone Number property. In this case, the default value might mean something different than if no value were set. With MTP 1.0, the initiator cannot distinguish between a property value that has been set as null and one that has not yet been set. To solve this issue, we enhanced GetObjectPropList to distinguish between properties that have default values and properties that have no initialized value.

For GetObjectPropList operations, a device should not return “Uninitialized properties” or default values as part of the GetObjectPropList result set. If the property is absent, the device should either omit it from the resulting dataset or return the property with the data type set to “Unknown”. The handling of unspecified properties includes the following semantics:

* Properties with a data type of “Unknown” are treated as if they had been omitted from the dataset.
* If an initiator requests all properties, the responder must return only the properties that have initialized values. If all properties are available and no properties are initialized, the responder returns only the count of properties as part of the data phase and the count must be 0.
* If uninitialized properties are returned as part of the data set with the data type of the property set to “Unknown”, the count of returned properties includes the uninitialized properties. For example, if the resulting property set contains five properties, three that have values and two uninitialized properties that have a datatype of “Unknown”, the number of properties that are specified in the dataset is still five.
* If an initiator requests a specific property and that property is currently uninitialized, the resulting dataset is identical to the all-properties dataset that has no property value (the second bullet in this list ) or contains a single property that has the data type specified as “Unknown” (the third bullet in this list).

MTP initiators should handle uninitialized properties for all object property requests regardless of whether the properties are associated with services. The complete handling of uninitialized properties simplifies property management on the client MTP device.

For GetObjectPropValue operations, which are supported only with classic MTP objects in classic storages, compatibility can be maintained by requiring that the default value be returned.

## 3.3 DeleteObjectPropList

The new DeleteObjectPropList operation enables an object property to be “deleted,” that is, to reset an object property to an uninitialized state. Device service initiators should use the DeleteObjectPropList operation to “delete” an object property instead of using SetObjectPropList with the default value specified.

DeleteObjectPropList is supported for all objects regardless of whether they are bound to a service or to a classic MTP storage.

## 3.4 SetObjectPropList/SendObjectPropList

The updated SetObjectPropList and SendObjectPropList operations include support for uninitialized properties. When encoding property sets, initiators should apply the same rules that were defined for [GetObjectPropList](#_GetObjectPropList).

When receiving these datasets, responders should treat missing properties or properties that are encoded as an “Unknown” type as uninitialized by the initiator. Responders must ignore these properties, and they must not delete them. For information on how to reset properties to their uninitialized state, see “[DeleteObjectPropList](#_4.2.8_DeleteObjectPropList)” earlier in this specification. Devices should not simply initialize these properties without default values. If a particular value is calculated (such as the object name) and the initiator does not provide a value, the responder can provide the value.

## 3.5 UpdateObjectPropList

The new UpdateObjectPropList operation enables an initiator to update the binary data of existing objects directly on the responder without first deleting and re‑creating the object. The UpdateObjectPropList operation must always be followed by a SendObject operation.

To modify only properties and not data, use SetObjectPropList instead of SendObjectPropList and SendObject.

The behavior of UpdateObjectPropList is identical to SendObjectPropList plus SendObject, except that UpdateObjectPropList modifies an existing object that is specified in the parameter list instead of creating a new object.

UpdateObjectPropList supports datasets with undefined properties.

As with GetObjectPropList, SetObjectPropList, and SendObjectPropList, you can use UpdateObjectPropList with any object on the device regardless of whether that object is associated with a service or is in a classic MTP storage.

## 3.6 GetFormatCapabilities

The new GetFormatCapabilites operation improves the performance of querying a device for the supported object properties on formats that are associated with classic MTP storages (those formats that appear in the DeviceInfo dataset). Responders should implement this operation because it replaces multiple calls to the device with a single, more efficient call.

## 3.7 Maintaining Compatibility with MTP 1.0

The MTP device services extension was designed specifically to maintain compatibility with the MTP specification while it provides new functionality for new types of content. Whenever possible, this specification uses MTP 1.0 assets.

## 3.8 Maintaining Compatibility with PTP and Other non-Windows MTP Initiators

Devices that must report a basic DeviceInfo dataset to maintain compatibility with Picture Transfer Protocol (PTP) devices (such as printers) can update the DeviceInfo dataset when they receive session initiator information that was sent during driver initialization.

The Windows MTP initiator can handle DeviceInfo dataset changes, but the following constraints exist in the Windows 7 implementation:

* Devices can add support for MTP device services and network configuration extensions, but they must not remove support for these extensions if they are already specified.
* Multitransport devices, such as those that support the functional unique identifier (FUID) device property, must report the FUID device property and the services extension in the initial DeviceInfo dataset. The FUID property is ignored if it is not present in the initial DeviceInfo dataset.

# 4.0 MTP Device Services Extension Implementation

The following sections describe the MTP device services extension and provide guidelines for how to implement support for the extension in initiators and responders.

## 4.1 MTP Extension Identification

A device that supports the MTP device services extension must include the following string in the VendorExtensionDescription field of the MTP DeviceInfo dataset that the device returns in response to the MTP GetDeviceInfo operation.

|  |  |  |  |
| --- | --- | --- | --- |
| Datasetfield | Field order | Size | Value |
| MTPVendorExtensionDesc | 4 | Varies | “Microsoft.com/DeviceServices:1.0” |

For more information, see the MTP specification.

## 4.2 Extension Operations

The following table lists the operation codes that the initiator and responder must implement to effectively support the MTP device services extension.

|  |  |
| --- | --- |
| Operation code | Operation name |
| 0x9301 | GetServiceIDs |
| 0x9302 | GetServiceInfo |
| 0x9303 | GetServiceCapabilities |
| 0x9304 | GetServicePropDesc |
| 0x9305 | GetServicePropList |
| 0x9306 | SetServicePropList |
| 0x9307 | UpdateObjectPropList |
| 0x9308 | DeleteObjectPropList |
| 0x9309 | DeleteServicePropList |
| 0x930A | GetFormatCapabilities |

The following sections describe the extension operations.

### 4.2.1 GetServiceIDs

This operation returns an array of ServiceIDs.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9301 |
| Operation Parameter 1 | None |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | An array of ServiceIDs |
| Data Direction | R -> I |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Parameter\_Not\_Supported |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameters: No operation parameters. |
| Response: No response parameters. |

### 4.2.2 GetServiceInfo

This operation returns the ServiceInfo dataset for the service that the ServiceID identifies in the first parameter. The ServiceInfo dataset is defined in the next section.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9302 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [ServiceInfo](#_4.2.2.1_ServiceInfo_Dataset) dataset |
| Data Direction | R -> I |
| ResponseCode Options | OK, Invalid\_ServiceID  |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:ServiceID: The ServiceID of the service that is being queried. |
| Response: No response parameters. |

#### 4.2.2.1 ServiceInfo Dataset

The ServiceInfo dataset communicates the core identification and asset information of a service. It includes the core service information, service properties, supported formats, supported methods, events, and raw data. The following sections of this specification describe the sections of the ServiceInfo dataset.

#### 4.2.2.2 Core Service Information

This section of the ServiceInfo dataset describes the core properties of an MTP device service. These properties describe the service’s identity and the location of the service content. ServiceID and ServiceStorageID may be dynamic, depending on device implementation. However, they must remain constant within an MTP session. The device defines PersistentServiceID as a session-independent method for reconnecting to a particular service. The remaining properties are defined by the definition of the service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size (bytes) | Datatype | Description |
| ServiceID | 1 | 4 | UINT32 | The number that is used to enumerate and address services. |
| ServiceStorageID | 2 | 4 | UINT32 | The MTP StorageID that holds the content for a particular service. StorageID is formatted according to standard rules for MTP storage IDs. |
| PersistentServiceID | 3 | 16 | UINT128 | The persistent GUID for a particular service. The GUID persists for the life of the device and can be assigned at compile time. It enables the initiator to recognize returning services regardless of ServiceID. It must be nonzero. |
| ServiceVersion | 4 | 4 | UINT32 | The version of service implementation. The version for this specification is 100 (0x64). For each version greater than 100, sections can be added to the end of the ServiceInfo dataset, which preserves compatibility with legacy responders and initiators. |
| ServiceGUID | 5 | 16 | UINT128 | The GUID that represents the type of service. Multiple services on a device can have the same ServiceGUID. Instances of each service that share the same ServiceGUID must have different persistent service IDs. |
| ServiceName | 6 | DTS | String | The computer-readable name for this service. |
| ServiceFlags | 7 | 4 | UINT32 | The following flag is defined:0x00000001: abstract service |
| BaseServiceID | 8 | 4 | UINT32 | Reserved for future use. This value is set to 0 for this specification. |
| UsesServiceGUIDs | 9 | DTS | AUINT128 | An array of persistent service IDs that identify abstract services that this service uses. |

#### 4.2.2.3 Service Properties

The service properties section of the ServiceInfo dataset lists properties of the actual service, not of the objects or the functionality of the service. The service properties list is fixed and specified in the ServiceInfo dataset. The values of each service property can be retrieved and written through the GetServicePropList, SetServicePropList, and DeleteServicePropList operations. Properties are in PKeys.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size(bytes) | Data type | Description |
| NumServiceProperties | 10 |  4 | UINT32 | The number of service properties in the dataset. The value can be 0. |
| ServicePropCode1 |   |  2 | UINT16 | The service property code. |
| ServicePropPKeyNamespace1 |   | 16 | UINT128 | The namespace for the service property. |
| ServicePropPKeyID |   |  4 | UINT32 | The ID for the service property. |
| ServicePropertyName |   | DTS | String | The computer-readable name of the service property.  |
| Repeat the service properties datasets. |

#### 4.2.2.4 Object Formats

Similar to MTP formats, each new object format has GUID and Multipurpose Internet Mail Extensions (MIME) types to enable the definition of new object formats that are not covered in the MTP specification. A service must list all supported formats in this section of the dataset. For GUIDs and interface names for existing MTP 1.0 formats, see the *BridgeDeviceService.h* file, which is provided in the MTP Device Enabling Kit. For new formats that are not covered by the MTP specification 1.0, you should fill in each field appropriately.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size(bytes) | Data type | Description |
| NumFormats |   |  4 | UINT32 | The number formats in this dataset. The value can be 0. |
| FormatCode1 |   |  2 | UINT16 | The format code. The code can be an MTP specification format code or another unique format code. All format codes on a device must be unique. |
| FormatGUID1 |   | 16 | UINT128 | The GUID of the format. |
| FormatName1 |   | DTS | String | The computer-readable name for this format. |
| BaseFormatCode1 |   |  2 | UINT16 | Reserved for future use. This value is 0 for this specification. |
| MIMEType1 |   | DTS | String | Reserved for future use. For formats that represent binary objects with well-defined MIME types, we recommend that you include the MIME type in the service definition. |
| Repeat the format datasets. |

#### 4.2.2.5 Methods

Methods represent actions on a service or another object. The method object is itself a command that is executed based on the parameters for a device. Parameters are communicated in the SendObjectPropList command. Methods are defined within the context of a service and are specific only to that service.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size(bytes) | Data type | Description |
| NumMethods |   | 4 | UINT32 | The number of methods. This field can be 0. |
| MethodFormatCode1 |   | 2 | UINT16 | The ID of the method (FormatCode). |
| MethodGUID1 |   | 16 | UINT128 | The GUID for this method. |
| MethodName1 |   | DTS | String | The method name in computer-readable form. |
| ObjectAssociationFormatCode1 |   | 2 | UINT16 | The object format code with which this method is associated. This field can be 0 if no format applies, for example, if the method is associated directly with the service itself. |
| Repeat for each method. |

#### 4.2.2.6 Events

Events are defined within the context of a service and are specific only to that service. MTP 1.0 events are not required to be declared in this section of the dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size(bytes) | Datatype | Description |
| NumEvents |   |  4 | UINT32 | The number of events. The value can be 0. |
| Event Datacode1 |   |  2 | UINT16 | The ID of the event (EventCode). |
| EventGUID1 |   | 16 | UINT128 | The GUID of the event. |
| EventName1 |   | DTS | String | The event name in computer-readable form. |

#### 4.2.2.7 Data

This section is for binary data, and is optional. To determine whether a particular service uses the data section, refer to the individual service definition.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Datasetfield | Fieldorder | Size(bytes) | Datatype | Description |
| Data |   | DTS | AUINT8 | The data stream. The value can have a size of 0 (empty). |

### 4.2.3 GetServiceCapabilities

All object format and method format information is reported by using the GetServiceCapabilities operation. Object formats use the extended service object property form, whereas method formats use the extended service method parameter form. Object formats and method formats can be listed in any order in the ServiceCapabilityList dataset. Abstract services do not have a ServiceCapabilityList dataset. If the service does not have formats or methods, the responder returns Operation\_Not\_Supported.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9303 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | [FormatCode] |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [ServiceCapabilityList](#_ServiceCapabilityList_Dataset) dataset |
| Data Direction | R -> I |
| ResponseCode Options | OK, Invalid\_ServiceID, Invalid\_ObjectFormatCode, Operation\_Not\_Supported |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameters:ServiceID: The ServiceID that is being queried.FormatCode: An optional parameter. If this parameter is used, the device returns only the capabilities for the specified format for the service. If this parameter is not used, it contains the value 0x00000000 and the device returns the capabilities of all object formats and method formats for the specified service. |
| Response: No response parameters. |

#### 4.2.3.1 ServiceCapabilityList Dataset

This dataset is returned for the GetServiceCapabilities operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| NumFormats | 1 | 4 | UINT32 | The count of formats. The value must equal the total number of object formats and method formats. |
| FormatCode | 2 | 2 | UINT16 | The format code for the ObjectPropDesc and InterdependentPropDesc datasets in the remainder of this table.**Note:** Both method formats and object formats can be reported in this dataset. No special ordering is required. |
| NumObjectPropDesc | 3 | 4 | UINT32 | The number of ObjectPropDesc arrays in the ServiceCapabilityList dataset. The value can be 0. |
| ObjectPropDesc dataset | N | DTS | DTS | A list of each of the ObjectPropDesc datasets. Service objects contain extended information by using the object forms that are described in “[Object Property Forms](#_4.6.1_Object_Property)” later in this specification.  |
| InterdependentPropDesc dataset |  | DTS | DTS | The dataset that is to be used as described in the InterdependentPropDesc dataset in the MTP specification. This field is 0 for method formats. |
| Repeat for each format/method. |

### 4.2.4 GetServicePropDesc

This operation returns theServicePropertyDesc dataset for the service that is identified by the ServiceID in the first parameter. If the service has no properties, this operation returns an empty dataset.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9304 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | [ServicePropCode] |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [ServicePropertyDesc](#_4.2.4.1_ServicePropertyDesc_Dataset) dataset |
| Data Direction | R -> I |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Parameter\_Not\_Supported, Invalid\_ServiceID, Invalid\_ServicePropCode |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameters:ServiceID: The ServiceID that is being queried.ServicePropCode: An optional parameter. If this parameter is specified, the device returns ServicePropertyDesc only for the specified service property ID. If this parameter is unused, the value is 0 and all service properties are returned.  |
| Response: No response parameters. |

#### 4.2.4.1 ServicePropertyDesc Dataset

The following table shows the dataset that is returned from the GetServiceProperties operation. If the service has no service properties, this dataset contains a value of 0x00000000 for NumProperties.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| NumProperties | 1 | 4 | UINT32 | The number of properties that apply to the service. |
| ServicePropertyCode | 2 | 2 | UINT16 | The service property code. |
| Datatype1 | 3 | 2 | UINT16 | The datatype code of the property. |
| Get/Set1 | 4 | 1 | UINT8 | Indicates whether the property is read-only (Get), or read-write (Get/Set):* 0x00 Get
* 0x01 Get/Set
 |
| Form Flag1 | 5 | 1 | UINT8 | The format of the next field:* 0x00 None
* 0x01 Range form
* 0x02 Enumeration form
* 0x03 DateTime form
* 0x04 Fixed-length Array form
* 0x05 Regular Expression form
* 0x06 ByteArray form
* 0xFF LongString form
* 0x83 Object ID
 |
| FORM | N/A | <variable> | - | This dataset depends on the form flag and is absent if Form Flag = 0x00. For the ObjectPropDesc dataset, see the MTP specification. |
| Repeat the dataset for each service property. |

### 4.2.5 GetServicePropList

This operation is similar to GetObjectPropList in the MTP specification, Revision 1.0. GetServicePropList reads properties from a service. Properties can be retrieved one at a time or all at once. This operation returns the ServicePropList dataset for the service that is identifiedbytheServiceID in the first parameter. See “[ServicePropList Dataset](#_ServicePropList_Dataset)” later in this specification.

If a ServiceProperty value has not been set (and is not the default value), a call to GetServicePropList should return only properties that have set values. If a single ServicePropCode is specified and that value has not yet been set (it is an “Uninitialized property”), the device should return a ServicePropList of size 0 or specify the data type of the service property as “Unknown”. This operation does not apply to abstract services. Abstract service property values can be obtained through the implementing service.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9305 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | [ServicePropCode] |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [ServicePropList](#_ServicePropList_Dataset) dataset |
| Data Direction | R -> I |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Parameter\_Not\_Supported, Invalid\_ServiceID, Invalid\_ServicePropCode |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameters:ServiceID: The ServiceID that is being queried.ServicePropCode: An optional parameter. If this parameter is specified, the device returns the ServicePropList dataset with only the single specified property value. If this parameter is set to 0x0, the device returns all ServiceProperty values for the specified service. |
| Response: No response parameters. |

#### 4.2.5.1 ServicePropList Dataset

The ServicePropList dataset lists field names and their data types.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Description |
| NumberOfElements | 1 | 4 | UINT32 | The count of property quadruples in this dataset. |
| Element1ServiceID | 2 | 4 | ObjectHandle | The ServiceID of the object to which Property1 applies. |
| Element1PropertyCode | 3 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes Property1. |
| Element1Datatype | 4 | 2 | Datacode | The DatatypeCode of Property1. |
| Element1Value | 5 | DTS | DTS | The value of Property1. |
| Element2ServiceID | 6 | 4 | ObjectHandle | The ServiceID of the object to which Property2 applies. |
| Element2PropertyCode | 7 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes Property2. |
| Element2Datatype | 8 | 2 | Datacode | The DatatypeCode of Property2. |
| Element2Value | 9 | DTS | DTS | The value of Property2. |
| ... |
| ElementNServiceID | 4\*N-2 | 4 | ObjectHandle | The ServiceID of the object to which PropertyN applies. |
| ElementNPropertyCode | 4\*N-1 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes PropertyN. |
| ElementNDatatype | 4\*N | 2 | Datacode | The DatatypeCode of PropertyN. |
| ElementNValue | 4\*N+1 | DTS | DTS | The value of PropertyN. |

### 4.2.6 SetServicePropList

This operation is similar to SetObjectPropList in the MTP 1.0 specification. This operation sets a ServiceProperty by using the ServicePropList dataset. It enables the writing of property values to a service. Properties can be written one at a time or all at once. This operation does not apply to abstract services. Abstract service property values must be set through the implementing service.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9306 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [ServicePropList](#_ServicePropList_Dataset) dataset |
| Data Direction | I->R |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Access\_denied, Property\_Not\_Supported, Device\_Busy, Store\_Not\_Available, Store\_Full, Invalid\_ServiceID, Invalid\_Dataset, Invalid\_ServicePropCode |
| Response Parameter 1 | [FailureCode] |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:ServiceID: The ServiceID for which properties are being written. |
| Response:FailureCode: If this operation fails and all ServiceProperty values that are sent in ServicePropList are not applied successfully, the operation must return a valid ResponseCode and must specify the zero-based index of the first property that was not applied in this parameter. The responder must not process any ServiceProperty values that follow the property that failed to update. If no ServiceProperty value was set, the initiator must return a FailureCode of 0x00000000. |

### 4.2.7 UpdateObjectPropList

This operation sets the property list for a particular object that will be updated with a new binary object. This operation can be used to replace the binary data of an existing object. An initiator uses this operation to send new metadata for an existing ObjectID and follows this operation with a SendObject operation for the same ObjectID. If this operation sends a property, the responder overwrites the existing property. The responder must maintain any existing properties for an object for which no new value is sent.

If this operation is not immediately followed by a SendObject operation, the responder must discard any property changes that are sent through this operation and retain the original object properties and object binary. If the following SendObject operation fails for any reason, the initiator can continue to issue SendObject operations for the current object handle until SendObject is successful or another operation is sent. If no successful SendObject occurs, the responder must discard all properties that were sent in this operation.

The dataset that is sent in this operation is similar to the ObjectPropList dataset that is sent and received in the SetObjectPropList and GetObjectPropList operations, respectively, but has additional restrictions on the values of the fields. All ObjectHandle fields must contain either the value 0x00000000 or must match the ObjectHandle that is specified in parameter 1. All properties that are defined in this operation will be applied to the object, which is sent in a subsequent SendObject operation. If any properties are inconsistent (that is, the property is not supported or the value is inconsistent for the existing ObjectFormat), the responder must fail this operation with the appropriate response code and indicate the zero-based index of the first failed property in the first return parameter.

If the object size in the data phase of this operation indicates that the object that is to be sent has a size of 0, an OK response suggests that the object has been sent successfully and a SendObject operation is not required. However, if a SendObject operation follows that contains an object of size 0, the responder does not fail the request.

The use of UpdateObjectPropList is not restricted to objects in services. It can also be used with objects in classic MTP storage.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9307 |
| Operation Parameter 1 | ObjectHandle |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | ObjectPropList dataset |
| Data Direction | I->R |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Access\_Denied, Store\_Read\_Only, Object\_Too\_Large, Store\_Full, Invalid\_ObjectFormatCode, Store\_Not\_Available, Parameter\_Not\_Supported, Invalid\_ObjectHandle, Invalid\_Dataset |
| Response Parameter 1 | [Index of failed property] |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:ObjectHandle: The object handle of the target object. |
| Response:Index of the failed property. If this operation fails, and all object property values that are sent in the ObjectPropList dataset are not applied successfully, the operation must return a valid ResponseCode and must indicate the zero-based index of the property that was invalid. The responder must not apply any object property value. If no object property value was set, the parameter must contain a value of 0x00000000. |

### 4.2.8 DeleteObjectPropList

This operation removes the properties that are specified in the DeleteObjectPropList dataset from the specified object or objects. If a property is not removable, the responder must set the property value to its default value. Only properties that are marked as writeable can be specified.

The first parameter is required and defines the object or objects for which properties are to be deleted. A value of 0xFFFFFFFF indicates that the object handles are specified in the dataset.

The dataset contains a list of ObjectHandle/PropertyCode pairs. If parameter 1 specifies a valid object handle, all ObjectHandle fields of the dataset must contain either the value 0x00000000 or must match the ObjectHandle that is specified in parameter 1. If parameter 1 specifies 0xFFFFFFFF, each ObjectHandle field in the dataset must specify a valid object handle. The dataset in this latter case can specify properties for multiple objects.

The responder applies all properties that are specified in this operation to the specified object or objects. If any properties are inconsistent (that is, the property is not supported or cannot be written), the responder fails this operation with the appropriate response code (Invalid\_ObjectHandle, Invalid\_ObjectPropCode, or Access\_Denied) and specifies the zero-based index of the first failed property in the first return parameter. All properties that are successfully applied before the failed property must remain deleted (or reset). Any properties after the failed property must remain unchanged.

The use of DeleteObjectPropList is not restricted to objects in services. It can also be used with objects in classic MTP storages.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9308 |
| Operation Parameter 1 | ObjectHandle.  |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [DeleteObjectPropList](#_4.2.8.1_DeleteObjectPropList_Datase) dataset |
| Data Direction | I->R |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Invalid\_ObjectHandle, Invalid\_ObjectPropCode, Access\_Denied, Store\_Read\_Only, Store\_Not\_Available, Parameter\_Not\_Supported, Invalid\_Dataset |
| Response Parameter 1 | [Index of failed property] |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:ObjectHandle: The ObjectID of the target object. This parameter is required and defines the object for which properties are to be deleted. A value of 0xFFFFFFFF indicates that the objects that are to be updated are specified in the DeleteObjectPropList dataset. |
| Response:Index of failed property: If this operation fails and not all object property values that are sent in the DeleteObjectPropList dataset are deleted (or reset) successfully, the operation must return a valid ResponseCode (Invalid\_ObjectHandle, Invalid\_ObjectPropCode, or Access\_Denied) and must indicate the zero-based index of the first property that was invalid. If no object property value was deleted (or reset), the parameter must contain a value of 0x00000000. |

#### 4.2.8.1 DeleteObjectPropList Dataset

The DeleteObjectPropList dataset contains the following fields.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Description |
| NumberOfElements | 1 | 4 | UINT32 | The count of ObjectHandle-PropertyCode pairs in this dataset. |
| Element1ObjectHandle | 2 | 4 | ObjectHandle | The ObjectHandle of the object to which Property1 applies. |
| Element1PropertyCode | 3 | 2 | Datacode | The datacode that identifies the ObjectPropDesc that describes Property1. |
| Element2ObjectHandle | 4 | 4 | ObjectHandle | The ObjectHandle of the object to which Property2 applies. |
| Element2PropertyCode | 5 | 2 | Datacode | The datacode that identifies the ObjectPropDesc that describes Property2. |
| ... |
| ElementNObjectHandle | 2\*N | 4 | ObjectHandle | The ObjectHandle of the object to which PropertyN applies. |
| ElementNPropertyCode | 2\*N+1 | 2 | Datacode | The datacode that identifies the ObjectPropDesc that describes PropertyN. |

### 4.2.9 DeleteServicePropList

This operation removes the properties that are specified in the DeleteServicePropList dataset from the specified service. If a property is not removable, it is returned to its default value. Only properties that are marked as writeable can be specified.

The first parameter is required, and it specifies the service for which properties are to be deleted.

The dataset contains a list of ServiceID-Property pairs. The ServiceID fields of the dataset must match the ServiceID that is specified in parameter 1.

All properties that are specified in this operation are applied to the specified service. If any properties are inconsistent (that is, the property is not supported or cannot be written), this operation must fail with the appropriate response code (Invalid\_ServiceID, Invalid\_ServicePropCode, or Access\_Denied) and must indicate the zero-based index of the first failed property in the first return parameter. All properties that are successfully applied before the failed property must remain deleted (or reset). Any properties after the failed property must remain unchanged.

This operation does not apply to abstract services. Abstract service property values must be deleted through the implementing service.

|  |  |
| --- | --- |
|  |  |
| Operation code | 0x9309 |
| Operation Parameter 1 | ServiceID |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [DeleteServicePropList](#_4.2.9.1_DeleteServicePropList_Datas) dataset |
| Data Direction | I->R |
| ResponseCode Options | OK, Operation\_Not\_Supported, Session\_Not\_Open, Invalid\_TransactionID, Access\_Denied, Store\_Read\_Only, Store\_Not\_Available, Parameter\_Not\_Supported, Invalid\_ServiceID, Invalid\_Dataset, Invalid\_ServicePropCode |
| Response Parameter 1 | [Index of failed property] |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:ServiceID: The ServiceID of the target object. This parameter is required and specifies the service for which properties will be deleted. |
| Response:Index of failed property: If this operation fails and not all service property values that are sent in the DeleteServicePropList dataset are deleted (or reset) successfully, the operation must return a valid ResponseCode (Invalid\_ServiceID, Invalid\_ServicePropCode, or Access\_Denied) and must indicate the zero-based index of the property that was invalid. If no object property value was deleted (or reset), the parameter must contain a value of 0x00000000. |

#### 4.2.9.1 DeleteServicePropList Dataset

The DeleteServicePropList dataset contains the following fields.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Description |
| NumberOfElements | 1 | 4 | UINT32 | The count of ServiceID-PropertyCode pairs in this dataset. |
| Element1ServiceID | 2 | 4 | ObjectHandle | The ServiceID of the object to which Property1 applies. |
| Element1PropertyCode | 3 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes Property1. |
| Element2ServiceID | 4 | 4 | ObjectHandle | The ServiceID of the object to which Property2 applies. |
| Element2PropertyCode | 5 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes Property2. |
| ... |
| ElementNServiceID | 2\*N | 4 | ObjectHandle | The ServiceID of the object to which PropertyN applies. |
| ElementNPropertyCode | 2\*N+1 | 2 | Datacode | The datacode that identifies the ServicePropDesc that describes PropertyN. |

### 4.2.10 GetFormatCapabilities

A responder must report all object format information by using the GetFormatCapabilities operation. This operation is a bulk operation that duplicates the functionality of GetObjectPropsSupported, GetObjectPropDesc, and GetInterdependentPropDesc. Devices should implement this operation in addition to the existing operations for greatest speed and compatibility. This operation does not apply to device services, which must use the GetServiceCapabilties operation to enumerate service object formats and methods.

|  |  |
| --- | --- |
| Operation code | 0x930A |
| Operation Parameter 1 | [FormatCode] |
| Operation Parameter 2 | None |
| Operation Parameter 3 | None |
| Operation Parameter 4 | None |
| Operation Parameter 5 | None |
| Data | [FormatCapabilityList](#_4.2.10.1__FormatCapabilityList) dataset |
| Data Direction | R -> I |
| ResponseCode Options | OK, Invalid\_ObjectFormatCode |
| Response Parameter 1 | None |
| Response Parameter 2 | None |
| Response Parameter 3 | None |
| Response Parameter 4 | None |
| Response Parameter 5 | None |
| Parameter:FormatCode: An optional parameter. If this parameter is used, the device returns only the capabilities for the specified format. If this parameter is not used, it must contain a value of 0x00000000 and the device must return the capabilities of all object formats for the device. |
| Response: No response parameters. |

#### 4.2.10.1 FormatCapabilityList Dataset

This dataset is returned for the GetFormatCapabilities operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| NumFormats | 1 | 4 | UINT32 | The count of formats, which should include all formats in the DeviceInfo dataset. |
| FormatCode | 2 | 2 | UINT16 | The format code for the ObjectPropDesc and InterdependentPropDesc datasets in the remainder of this table. |
| NumObjectPropDesc | 3 | 4 | UINT32 | The number of ObjectPropDesc arrays to follow. |
| ObjectPropDesc dataset | N | DTS | DTS | List each of the ObjectPropDesc datasets.  |
| InterdependentPropDesc dataset |  | DTS | DTS | Use this dataset as described in the InterdependentPropDesc dataset in the MTP specification. |
| Repeat for each format. |

## 4.3 Extension Responses

The following table lists the extension response codes.

|  |  |
| --- | --- |
| Response code | Response name |
| 0xA301 | Invalid\_ServiceID |
| 0xA302 | Invalid\_ServicePropCode |

Invalid\_ServiceID

Response Code: 0xA301
Returned if the specified ServiceID does not exist or is an invalid value.

Invalid\_ServicePropCode

Response Code: 0xA302
Returned if the specified service property code does not exist or is an invalid value.

## 4.4 Extension Device Properties

The following table lists the device property codes that are associated with device property names.

|  |  |
| --- | --- |
| Device property code | Device property name |
| 0xD301 | Functional ID |
| 0xD302 | Model ID |
| 0xD303 | UseDeviceStage flag |

### 4.4.1 Functional ID

The functional ID is a 128-bit GUID that must be permanent for the life of the device and appear over all transports (USB, IP, and Bluetooth). The functional ID is used to uniquely identify an MTP device that is connected by using multiple transports (either simultaneously or one at a time).

The DefaultValue and CurrentValue fields can be set by the manufacturer. The Get/Set value must be 0x00 (GET). The UINT128 GUID must meet the following conditions:

* Be 100-percent unique across all devices from all manufacturers (a true GUID).
* Persist across all transports.
* Persist between MTP sessions and device resets.

If the values are left empty, an initiator should generate and set a compliant UINT128 GUID as the value. The device must persist this value across all transports, MTP sessions, and device resets. The Get/Set value must be 0x01 (GET/SET).

The following table shows the required properties of the functional ID.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Value |
| PropertyCode | 1 | 2 | UINT16 | 0xD301 |
| Datatype | 2 | 2 | UINT16 | 0x00A (UINT128) |
| Get/Set | 3 | 1 | UINT8 | Device-defined |
| DefaultValue | 4 | Varies | Varies | [DefaultValue] |
| CurrentValue | 5 | Varies | Varies | [CurrentValue] |
| FormFlag | 6 | 1 | UINT8 | 0x00 None |

### 4.4.2 Model ID

The model ID is a 128-bit GUID that uniquely identifies the model of the device. The model ID can be as specific or as generic as the manufacturer chooses. For example, the model ID can differ among product models, colors of an individual model, or even individual devices. An MTP initiator can use model IDs for many purposes, such as customizing the experience for a particular known device or identifying differences among devices where the model string in DeviceInfo is insufficient. Model IDs should be set by the manufacturer and are read-only.

The following table shows the required properties of a model ID.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Value |
| PropertyCode | 1 |  2 | UINT16 | 0xD302 |
| Datatype | 2 |  2 | UINT16 | 0x00A (UINT128) |
| Get/Set | 3 |  1 | UINT8 | 0x00 GET |
| DefaultValue | 4 | 16 | UINT128 | [DefaultValue] |
| CurrentValue | 5 | 16 | UINT128 | [CurrentValue] |
| FormFlag | 6 |  1 | UINT8 | 0x00 None |

### 4.4.3 UseDeviceStage Flag

This device property indicates that the device can be used with Windows Device Stage™. When the device is first plugged into Windows, Windows uses a “baseline/generic” Device Stage experience instead of AutoPlay and retrieves the customized Device Stage metadata at a later time from the Microsoft servers. This is primarily designed for devices that cannot store metadata on the device with the device metadata service but have metadata on the Microsoft servers. Devices that have no custom metadata can also use this device property to receive the “baseline” Device Stage experience instead of AutoPlay.

To enable this feature, the device must:

* Include the device property (0xD303) in the DeviceInfo dataset.
* Set the CurrentValue and DefaultValue of the property to 1.

**Note:** If the property is included on the device but the CurrentValue is set to 0, Device Stage is *not* enabled for the device. For Device Stage to be enabled, the value must be nonzero.

The following table shows the required properties of the UseDeviceStage Flag.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Datatype | Value |
| PropertyCode | 1 | 2 | UINT16 | 0xD303 |
| Datatype | 2 | 2 | UINT16 | 0x002 (UINT8) |
| Get/Set | 3 | 1 | UINT8 | 0x00 GET |
| DefaultValue | 4 | 1 | UINT8 | [DefaultValue] |
| CurrentValue | 5 | 1 | UINT8 | [CurrentValue] |
| FormFlag | 6 | 1 | UINT8 | 0x00 None |

## 4.5 Extension Events

The following table lists the extension event codes.

|  |  |
| --- | --- |
| Event code | Event name |
| 0xC301 | ServiceAdded |
| 0xC302 | ServiceRemoved |
| 0xC303 | ServicePropChanged |
| 0xC304 | MethodComplete |

### 4.5.1 ServiceAdded Event

This event indicates that a new service is available on the device. The ServiceID parameter indicates the ServiceID of the new service. If more than one service is added at the same time, ServiceID must be 0xFFFFFFFF.

Event Code: 0x C301

Parameter 1: ServiceID

Parameter 2: None

Parameter 3: None

### 4.5.2 ServiceRemoved Event

This event indicates that a service is no longer available on the device. The ServiceID parameter indicates the ServiceID of the removed service. If more than one service is removed at the same time, ServiceID must be 0xFFFFFFFF.

Event Code: 0x C302

Parameter 1: ServiceID

Parameter 2: None

Parameter 3: None

### 4.5.3 ServicePropChanged Event

This event indicates that a service property value has been changed for one of the services. The ServiceID field indicates which service has a changed property. ServicePropCode is the service property code that has a changed value. The responder must send one event for each service with a changed property. If multiple services change properties at the same time, events are sent one at a time. If multiple properties on a single service have changed at the same time, ServicePropCode must be 0xFFFFFFFF.

Event Code: 0xC303

Parameter 1: ServiceID

Parameter 2: ServicePropCode

Parameter 3: None

### 4.5.4 MethodComplete Event

This event indicates that the responder has completed executing a method. The initiator then inspects the method results and deletes the method object.

ServiceID represents the service ID of the service in which the method was executing.

ObjectID represents the object ID of the method that was executing.

ResponseCode contains the MTP ResponseCode result, which indicates the completion status of the method execution. Do not confuse this status with the result code of the method itself, which is retrieved from the object properties that are defined on the method object.

Event Code: 0xC304

Parameter 1: ServiceID

Parameter 2: ObjectID

Parameter 3: ResponseCode

## 4.6 Extension Object Properties

The MTP device services extension includes three new forms for the ObjectPropDesc dataset.

Important Note:  Only object properties within services can use these new forms. Legacy MTP storages must use standard MTP forms.

The following are the new forms and their associated flags:

Service Object Property Form (0x81)

All object properties on object formats in a service must use this form. This form extends the ObjectPropDesc dataset with service-related information (PKey and Name).

Service Method Parameter Form (0x82)

All parameters on method format codes in a service must use this form. This form extends the ObjectPropDesc dataset with service-related information (PKey and Name) and parameter information (Type and Order).

Object ID Form (0x83)

This new form flag has no form (0x0). Object properties or method parameters that use the 0x83 form flag indicate that the UINT32 value of the property refers to an ObjectID (object handle).

### 4.6.1 Object Property Forms

For responders that implement this extension, object property datasets have three additional forms. The following table describes the extended dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| Property Code | 1 | 2 | UINT16 | The ObjectPropCode that identifies this property. ObjectPropCodes must be unique for each service. |
| Datatype | 2 | 2 | UINT16 | The datatype code of the property. |
| Get/Set | 3 | 1 | UINT8 | Indicates whether the property is read-only (Get), or read/write (Get/Set):* 0x00 Get
* 0x01 Get/Set
 |
| Default Value | 4 | DTS | DTS | The value of the factory default for the property. |
| Group Code | 5 | 4 | UINT32 | The retrieval group to which this property belongs. |
| Form Flag | 6 | 1 | UINT8 | Additional forms:* 0x81 Service Object Property form
* 0x82 Service Method Parameter form
* 0x83 Object ID form
 |
| FORM | N/A | Varies | Varies | This dataset depends on the form flag and is absent if Form Flag = 0x00. |

#### 4.6.1.1Service Object Property Extension Form

All object properties on object formats in a service must use this form. Standard MTP 1.0 forms follow the declaration of the PKeyNamespace, PKeyID, and PropertyName values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| Object Property PKeyNamespace | 7 | 16 | UINT128 | The PKey Namespace. |
| Object Property PKeyID | 8 | 4 | UINT32 | The PKey ID. |
| PropertyName | 9 | DTS | String | The computer-readable name for the object property. |
| Form Flag | 10 | 1 | UINT8 | The format of the next field:* 0x00 None
* 0x01 Range form
* 0x02 Enumeration form
* 0x03 DateTime form
* 0x04 Fixed-length Array form
* 0x05 Regular Expression form
* 0x06 ByteArray form
* 0xFF LongString form
* 0x83 Object ID form
 |
| FORM | N/A | varies | varies | This dataset depends on the form flag and is absent if Form Flag = 0x00. |

#### 4.6.1.2 Service Method Parameter Extension Form

All parameters on method format codes in a service must use the following form.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Fieldname | Field order | Size (bytes) | Data type | Description |
| Property PKeyNamespace | 7 | 16 | UINT128 | The PKey Namespace. |
| Property PKeyID | 8 | 4 | UINT32 | The PKey ID. |
| PropertyName | 9 | DTS | String | The IDispatch name. |
| ParameterType | 10 | 1 | UINT8 | * 0 if reserved
* 1 if in
* 2 if out
* 3 if in/out

GET/SET information must match the parameter type. Parameters of type 1 or 3 (in or in/out) must be GET/SET. Parameters of type 2 (out) must be GET. |
| ParameterNumber | 11 | 1 | UINT8 | Parameter number, used by methods. The result should be indicated as parameter 0 and labeled ”out” in ParameterType. |
| Form Flag | 12 | 1 | UINT8 | This field indicates the format of the next field:* 0x00 None
* 0x01 Range form
* 0x02 Enumeration form
* 0x03 DateTime form
* 0x04 Fixed-length Array form
* 0x05 Regular Expression form
* 0x06 ByteArray form
* 0xFF LongString form
* 0x83 Object ID form
 |
| FORM | N/A | Varies | Varies | This dataset depends on the form flag and is absent if Form Flag = 0x00. |

ParameterNumber must be set according to the definition of the method, beginning at 0. The ParameterType value must be set according to the type of parameter. The value for input parameters is 0x01, the value for output parameters is 0x02, and the value for in/out parameters is 0x03. Methods can contain multiple input and output parameters or can use none at all. The 0th output parameter must always contain the result for the method, if one exists.

# 5.0 Resources

USB Media Transfer Protocol Specification, Revision 1.0

<http://www.usb.org/developers/devclass_docs/MTP_1.0.zip>

MTP Device Services for Windows

<http://www.microsoft.com/whdc/device/wpd/MTP-DevServ_Win7.mspx>

Portable Device Enabling Kit for MTP, Version 7R2

<http://www.microsoft.com/whdc/device/wpd/MTP-DEK_Win7.mspx>

Windows Portable Devices

<http://www.microsoft.com/whdc/device/wpd/default.mspx>

Windows Portable Devices Team Blog

<http://blogs.msdn.com/wpdblog/default.aspx>